

Conjunctive Query Pattern Structures: A relational database model for Formal Concept Analysis

Jens Kötters and Peter W. Eklund*

*Centre for Cyber Security Research and Innovation
Deakin University, Geelong, Australia.

Abstract. Previous work on concept graphs combining Formal Concept Analysis with Conceptual Graphs is revisited in this paper in the context of conjunctive database queries. The paper presents the conjunctive database query model as a special case of Ganter and Kuznetsov’s pattern structures, where patterns are considered as conjunctive queries. It further presents kernel operators to reduce the complexity of the computation and to simplify the description of concepts. Alternative formalisms based on intensional graphs and power context families are introduced to advance knowledge interoperability between Formal Concept Analysis, database theory, model theory, logic, concept graphs and Sowa’s Conceptual Graphs.

Keywords: Conjunctive Queries, Pattern Structures, Power Context Families, Concept Graphs, Conceptual Graphs

1 Introduction

Consistent with the terminology and understanding of philosophers and linguists [30, 13, 29], a concept has an *extension* and an *intension*. In this understanding of a concept, we say that ‘extension’ refers to the things belonging to a concept, whereas ‘intension’ refers to the intended meaning of a concept.

Formal Concept Analysis (FCA) [16] provides a mathematical formalization of concepts which represent the extension by a set of formal objects (the *extent*) and the intension by a set of formal attributes (the *intent*). Formal Concept Analysis as a framework for analyzing and exploring data was developed at the Technical University in Darmstadt in the 1980’s [16] and has been used in a variety of applications such as medicine, psychology, social sciences, linguistics, information sciences, machine and civil engineering (see Groh et. al [19] for a survey of applications).

Baader and Molitor [5] established a variant of Formal Concept Analysis (FCA) where concept intents are expressed in a formal language, namely a description logic [3]. Their approach positions FCA closer to natural language, because it introduces an explicit treatment of syntactic structure and the ability to capture, not only properties of objects, but also relations between them. A

wide variety of description logics exist, these differ by language features and expressiveness. One relatively simple but natural description logic is the existential language $\mathcal{EL}$, where concept descriptions are built from atomic concepts using the operators top-concept ($\top$), conjunction ($\sqcap$) and existential restriction ($\exists$).

Concept descriptions in $\mathcal{EL}$ have structural representations as rooted trees, their entailment is described by homomorphisms [4]. As noted by Baader et al. [4, p. 98], such concept descriptions form a special case of conjunctive queries, which are more generally described by graphs. Conjunctive queries, their structural representation, and the characterization of their entailment by homomorphisms, are due to Chandra and Merlin [7]. A variant of Formal Concept Analysis where concept intents are conjunctive queries (or infinite generalizations of these) is presented by Kötters [21], the formalizations draw on Chandra and Merlin’s work [7]. Conjunctive queries are represented by windowed structures, tableau queries [1, p. 43f.] drawn as labeled graphs.

Compared to concept descriptions in $\mathcal{EL}$, which are represented by rooted trees, general conjunctive queries are based on classical logic, can express cycles in the data and can describe, not only sets (i.e. unary relations), but also binary and higher-arity relations. As a consequence of the latter, concept extents in Kötters’ work [21] are relations (tables in a relational database), rather than sets of formal objects. In this way, a theoretical basis for combining Formal Concept Analysis with relational database theory is established. We recreate these results in Sections 4 and 5, after presenting and elaborating on database theoretic fundamentals in Section 3.

Pattern structures by Ganter and Kuznetsov [15] provide a theoretical underpinning for concepts with non-standard intents, such as numeric intervals, sequences, or the aforementioned concept descriptions in a description logic. Ganter and Kuznetsov present their formalism with an example using molecular graphs as concept intents; these are labeled graphs, used to represent the structure of chemical compounds. They write [15, p. 129]:

“Molecular graphs are only one example where such an approach is natural. Another, perhaps even more promising domain is that of Conceptual Graphs (CGs) in the sense of Sowa [33] and hence, of logical formulas.”

Indeed, Sowa [33] states a translation of conceptual graphs to primitive positive formulas [33, p. 86], namely formulas built from atoms using only conjunction ($\wedge$) and existential quantification ($\exists$). This means that conceptual graphs (in their basic form) correspond to conjunctive queries. However, the conceptual model of Section 5 does not fully translate to pattern structures, in particular because concept extents are tables, whereas pattern structures require concept extents to be sets.

In this paper a meaningful set representation of extents (originally mentioned briefly in previous work [25, p. 55]) is presented and we formulate the approach for pattern structures in Section 6.1. Also in their introductory paper, Ganter and Kuznetsov [15] covered kernel operations (or projections) for the simplification of concept intents, and mentioned adaptations of Ganter’s “Next Concept” [14]

algorithm to compute concepts. We apply this to our own work in Sections 6.2 and 6.3, respectively.

Wille connected the disciplines of Formal Concept Analysis and Conceptual Graphs with his seminal paper [36] on formalizing traditional philosophical logic following C.S. Peirce [30]. While some subsequent work reflects an understanding of conceptual graphs as conjunctive queries by Groh and Eklund [18], a summary of later developments by Wille himself [38, Section 6.5] does not reflect this understanding. The reason for this is that concept graphs, which are Wille's mathematization of conceptual graphs, fuse both data and description (thus combining the extensional and intensional perspectives), namely concept graphs are not separate from their extensional data. The preliminary notion of abstract concept graph [36, p. 300] still maintains a separation from data but has not received much attention in subsequent work. In Section 7, we present conjunctive queries as a type of abstract concept graphs (called intension graphs), and re-examine Wille's work in light of the preceding results.

Conjunctive queries, as a normalised representation for concept graphs, connect Formal Concept Analysis (FCA) to relational database theory and provide a mechanism through which FCA can scale using established relational database tools enabling their use as a knowledge representation for applications involving big data. Further, conjunctive queries are a more suitable input format for automated theorem provers [28].

2 Preliminaries

In order to make the connections between database theory and Formal Concept Analysis explicit, it is important to introduce the reader the terminology of logic and database theory. We therefore present some basic definitions used in relational algebra as follows.

A *relational signature* is a set M of relation symbols, each with an associated arity $n \geq 1$. We denote by M_n the subset of M that contains all symbols of arity n . An *atom* over M is a formula $Rx_1 \dots x_n$ or $x_1 = x_2$, where $n \geq 1$, $R \in M_n$, and $x_1, x_2, \dots, x_n$ are taken from a countably infinite set **var** of variables. An *M-formula* is a first-order formula built from atoms over M . The set of free variables occurring in a formula φ is denoted $\text{free}(\varphi)$.

A *relational structure* over M (or *M-structure*) is a pair $\mathfrak{A} := (A, (R_m)_{m \in M})$ consisting of a *carrier set* A and an n -ary relation $R_m \subseteq A^n$ for each $m \in M_n$ and $n \geq 1$. We set $|\mathfrak{A}| := A$, and $m^{\mathfrak{A}} := R_m$ for $m \in M$. We say that $\mathfrak{A}$ is *finite* if both $|\mathfrak{A}|$ and $\{m \in M \mid m^{\mathfrak{A}} \neq \emptyset\}$ are finite. The *empty M-structure* is the unique M -structure $\mathbf{0}_M$ with $|\mathbf{0}_M| = \emptyset$. A homomorphism $f : \mathfrak{A} \rightarrow \mathfrak{B}$ of M -structures is a function $f : |\mathfrak{A}| \rightarrow |\mathfrak{B}|$ such that $(f(a_1), \dots, f(a_n)) \in m^{\mathfrak{B}}$ for all $(a_1, \dots, a_n) \in m^{\mathfrak{A}}$, $m \in M_n$ and $n \geq 1$.

A *variable assignment* in an M -structure Δ is a function $\mu : \mathbf{var} \rightarrow |\Delta|$. We call the pair (Δ, μ) an *M-interpretation* (cf. [10]). Given an M -formula φ , we say that (Δ, μ) *satisfies* φ , written $(\Delta, \mu) \models \varphi$, if φ holds in Δ under the assignment μ (for details see [10]). A formula is called *satisfiable* if it is satisfied by some

interpretation. Given M -formulas φ and ψ , we say that φ *logically implies* ψ , written $\varphi \models \psi$, if every M -interpretation that satisfies φ also satisfies ψ . The formulas φ and ψ are called *logically equivalent* if $\varphi \models \psi$ and $\psi \models \varphi$.

A *partial function* $f : X \rightarrow Y$ is a function $f : D \rightarrow Y$ defined on a subset $D \subseteq X$. Its *domain of definition* is thus $\text{def}(f) := D$, and its *range* is $\text{rng}(f) := \{f(a) \mid a \in D\}$. For $U \subseteq X$, the *restriction* $f|_U$ is the partial function with $\text{def}(f|_U) = \text{def}(f) \cap U$ and $f|_U(a) = f(a)$ for $a \in \text{def}(f|_U)$. Given $f : X \rightarrow Y$ and $g : X \rightarrow Y$, we write $f \leq g$ to indicate that $f = g|_{\text{def}(f)}$.

Some results in this paper require dealing with proper classes and infinite cardinals, the necessary background can be found in the textbook of Ebbinghaus [9]. In particular, brackets as in $[x \mid P(x)]$ are used to denote classes. Otherwise, we use the same terminology for classes (e.g. functions, relations ...) that is also used for sets.

2.1 Formal Concept Analysis

In the early 1980s, Rudolf Wille presented a mathematization of the traditional philosophical understanding of concepts [34]. This was the foundation of a new research discipline, Formal Concept Analysis (FCA) [16]. The main characteristics of this approach are the formalization of extension and intension (which gives concepts a structure), and the origination of concepts from a context. In standard FCA, a context is a triple (G, M, I) , more specifically called a *formal context*, which consists of a set G of *objects*, a set M of *attributes*, and an *incidence relation* $I \subseteq G \times M$. A formal context is typically represented by a cross table, such as the one in Fig. 1. The row and column headers represent the objects and attributes, respectively, and a cross indicates that an object has an attribute (these are the object-attribute pairs in I).

	male	female	parent
Anne		×	×
Bob	×		×
Chris	×		
Dora		×	

Fig. 1. Formal context.

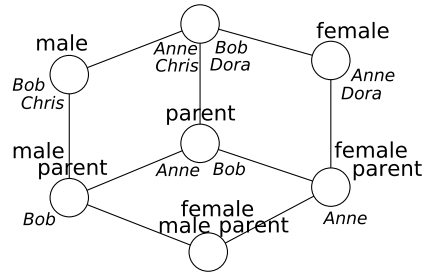


Fig. 2. Concept lattice.

Two operations are defined on a formal context, given by

$$B' := \{g \in G \mid (g, m) \in I \text{ for all } m \in B\}, \quad (1)$$

$$A' := \{m \in M \mid (g, m) \in I \text{ for all } g \in A\} \quad (2)$$

for all $B \subseteq M$ and $A \subseteq G$. The set B' contains the objects which have all attributes in B , and A' contains the attributes shared by the objects in A . An early application of FCA describes document retrieval by keywords [17]. In this case, B in (1) is considered a query (a set of keywords), and B' is the result set (a set of documents). The operations (1) and (2) form a Galois connection (cf. [16] or Section 5). For one-element sets $\{x\}$, it is usual to write x' instead of $\{x\}'$ in (1) and (2).

A *concept* of the context (G, M, I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. We say that A is the *extent*, and B is the *intent*, of the concept (A, B) . Seven concepts originate from the example context in Fig. 1, and these are shown in Fig. 2, arranged in a Hasse diagram by the order on their extents (or alternatively, the dual order on their intents). We can name the first six concepts, in reading order, *male*, *person*, *female*, *father*, *parent* and *mother*; the labels below and above each concept in Fig. 2 represent its extent and intent, respectively. The ordered set of concepts is denoted by $\underline{\mathcal{B}}(\mathbb{K})$, where $\mathbb{K} = (G, M, I)$ denotes the context. For further information on standard FCA, the reader is referred to [16].

2.2 Pattern structures

Pattern structures, introduced by Ganter and Kuznetsov [15], provide a foundation for concepts with intents other than sets of attributes [16]. Intents are taken from a meet-semilattice $(D, \sqcap)$, and are compared by the order $c \sqsubseteq d :\Leftrightarrow c = c \sqcap d$, which is the unique order on D with infimum $\sqcap$. An element of D is called a *pattern*, and $c \sqsubseteq d$ means that the pattern c is more generic than (or equal to) the pattern d .

Formally, a *pattern structure* is a triple $(E, (D, \sqcap), \delta)$ consisting of a set E of objects, a meet-semilattice $(D, \sqcap)$ of patterns and a function $\delta : E \rightarrow D$ describing each object by a pattern, where the infimum $\bigcap_{e \in A} \delta(e)$ must exist for all subsets $A \subseteq E$. The Galois connection in [15] is stated as

$$d^\square := \{e \in E \mid d \sqsubseteq \delta(e)\}, \quad (3)$$

$$A^\square := \bigcap_{e \in A} \delta(e) \quad (4)$$

for $d \in D$ and $A \subseteq E$. We may write the infimum in (4) as $\bigcap \delta(A)$, where $\delta(A) := \{\delta(e) \mid e \in A\}$. A *concept* of $(E, (D, \sqcap), \delta)$ is a pair (A, d) with $d^\square = A$ and $A^\square = d$. The concepts obtained from a pattern structure are called *pattern concepts*. They are ordered by their extents (or dually by their intents) and form a complete lattice. We only state their infimum in Prop. 1, which will be used later. For further information on pattern structures, the reader is referred to [15].

Proposition 1. *Let $\mathbb{S} := (E, (D, \sqcap), \delta)$ be a pattern structure, and $(A_i, d_i)_{i \in I}$ a family of concepts of $\mathbb{S}$. Their infimum is the concept $(\bigcap_{i \in I} A_i, \bigcap \delta(\bigcap_{i \in I} A_i))$.*

Proof. For brevity, we set $S := \bigcap_{i \in I} A_i$. We have to show that $(S, \bigcap \delta(S))$ is a concept; the infimum property is then clear from the order on extents. By (4),

$S^\square = \bigcap \delta(S)$. By (4), the other condition $(\bigcap \delta(S))^\square = S$ means that

$$\bigcap \delta(S) \leq \delta(e) \Leftrightarrow e \in S \quad (5)$$

for all $e \in E$, cf. (4). We assume $\bigcap \delta(S) \leq \delta(e)$. Then for all $i \in I$, we have $d_i = A_i^\square = \bigcap \delta(A_i) \leq \bigcap \delta(S) \leq \delta(e)$, and thus $e \in d_i^\square = A_i$ which shows $e \in S$. The other direction in (5) is immediate. $\square$

3 Conjunctive Queries: Basic Theorems

As Abiteboul et al. state [1, p. 8], conjunctive queries “*form the core of the select-from-where clause of the standard language in database systems, SQL, and are perhaps the most important class of [database] queries from a practical standpoint*”. In the form of SQL queries, conjunctive queries are thus commonly encountered in practice, but they come in a variety of other forms as well (cf. [1, Chpt. 4]). In their seminal paper [7], Chandra and Merlin introduce conjunctive queries essentially as certain logical formulas, and then represent them by relational structures (which correspond to tableau queries, as defined in Abiteboul et al. [1]), to prove what became two fundamental results of database theory: the homomorphism theorem and query minimization.

In this section, we reproduce the results of Chandra and Merlin [7] in the context of our work. Some variations of conjunctive queries are considered in the literature; in Section 3.1, we set the basic definitions that apply to this paper, and show how conjunctive queries can be drawn as graphs. Tableau queries, and their equivalence to logic-based conjunctive queries, are covered in Section 3.2. Section 3.3 introduces windowed structures and their homomorphisms [21] as infinite generalizations of tableau queries. The homomorphism theorem is usually formulated for tableau queries in the same free variables; it then says that homomorphisms express logical implication (in the opposite direction). In Section 3.4, we formulate the homomorphism theorem for arbitrary tableau queries, resorting to a variant of logical implication that deals with undefined variables. This answers, by extension, the question of what windowed structure homomorphisms mean. In Section 3.5, we introduce an order on tables in which the well-known natural join is the infimum operation. We state a set representation of tables, which reveals what comparison in the table order means. In Section 3.6, we restate the homomorphism theorem analogue to the version in Abiteboul et al. [1, p. 115], characterizing homomorphisms in terms of the table order. Query minimization, covered in Section 3.7, is useful to simplify queries in drawings and practical computations, but is not fundamental to the subsequent theory. We state the well-known result for tableau queries.

3.1 Queries, Data and Results

Conjunctive queries are introduced by Chandra and Merlin [7] as syntactic expressions $(x_1, \dots, x_k). \varphi$, where φ is a formula

$$\exists z_1 \dots \exists z_\ell (A_1 \wedge \dots \wedge A_m) \quad (6)$$

over a finite relational signature, $A_1, \dots, A_m$ are atoms of the form $R_i y_1 \dots y_{n_i}$, and $x_1, \dots, x_k$ are distinct variables with $\text{free}(\varphi) \subseteq \{x_1, \dots, x_k\}$. In the same paper, a *relational database* is represented as a finite, non-empty relational structure B . The result of $(x_1, \dots, x_k). \varphi$ in B is defined in [7] as the tuple set $\{(\mu(x_1), \dots, \mu(x_k)) \mid (B, \mu) \models \varphi\}$. This set corresponds to a k -column table, and its elements represent the table rows. Every formula built from atoms using only conjunction and existential quantification (such a formula is called *primitive positive*) can be equivalently represented as in (6), and is then said to be in *prenex normal form*.

In this paper we allow both the signature and the relational structure, which represents the data, to be infinite. We use the letters M , Δ and G to represent the signature, the data, and the carrier set of Δ , respectively. We consider conjunctive queries with equality, i.e. atoms of the form $x_j = x_k$ in (6) are allowed. We only consider conjunctive queries without constants. This does not limit expressivity (constants can be introduced as unary relations), and we avoid the special treatment of constants in database theory which can lead to unsatisfiable queries in the presence of equality (cf. [1, p. 47f.]). For a query $(x_1, \dots, x_k). \varphi$, we require $\text{free}(\varphi) = \{x_1, \dots, x_k\}$ (instead of $\text{free}(\varphi) \subseteq \{x_1, \dots, x_k\}$ as above), which does not limit expressivity, because trivial equalities $x_i = x_i$ can be added to φ without changing the query. The prefix $(x_1, \dots, x_k)$ then only specifies column order in the result. We abstract from column order by representing results in the *named perspective* [1, p. 31f.], where table rows are modeled as functions from column names to elements of G . Moreover, we use the free variables themselves as column names. As a result of the above specifications, a conjunctive query can be identified with a primitive positive formula. The chosen representation unifies the logical and database-theoretic viewpoints.

A *named tuple* is a partial function $\lambda : \mathbf{var} \rightarrow G$. For given $X \subseteq \mathbf{var}$, an X -*tuple* is a named tuple λ with $\text{def}(\lambda) = X$. So $\mathfrak{P}(G^X)$ is the set of all X -tuples, and $\text{Tup}(G) := \bigcup_{X \subseteq \mathbf{var}} G^X$ denotes the set of all named tuples. A *table* over G is a set $A \subseteq G^X$ of X -tuples over G . The elements of A are called its *rows*, and if A is non-empty, then X is called the *table schema*. The table schema of the empty table is defined to be $\mathbf{var}$. We denote the set of all tables over G by Tab_G . The result of the conjunctive query φ in Δ is the table

$$\text{res}_\Delta(\varphi) := \{ \mu|_{\text{free}(\varphi)} \mid (\Delta, \mu) \models \varphi \}. \quad (7)$$

Different representations of conjunctive queries are presented in the “Alice book” [1]. A conjunctive query in the original sense of the definition is considered a *conjunctive calculus query*. Tableau queries, covered in Section 3.2, are a structural representation of conjunctive queries.

We can elaborate on the prenex normal form in (6), representing a conjunctive calculus query as a formula

$$\exists z_1 \dots \exists z_\ell \exists z_{\ell+1} \dots \exists z_{\ell+k} (x_1 = z_{\ell+1} \wedge \dots \wedge x_k = z_{\ell+k} \wedge B_1 \wedge \dots \wedge B_r), \quad (8)$$

where each x_i has exactly one free occurrence (on the left side of an equation) and each atom B_j is of the form $m_j y_{j1} \dots y_{jn_j}$ with $n_j \geq 1$, $m_j \in M_{n_j}$ and

$y_{j_1}, \dots, y_{j_{n_j}} \in \{z_1, \dots, z_{\ell+k}\}$. This formula is obtained from (6) by introducing a new bound variable $z_{\ell+i}$ and an equation $x_i = z_{\ell+i}$ for each x_i , replacing x_i with $z_{\ell+i}$ in $A_1 \wedge \dots \wedge A_m$, and eliminating all equality atoms $z_{i_1} = z_{i_2}$ by substituting z_{i_1} with z_{i_2} , hence $r \leq m$. As a result of the latter, the variables $z_{\ell+1}, \dots, z_{\ell+k}$ in (8) need not be distinct. The normal form in (8) separates the free variables, which are used as column names, from the other variables.

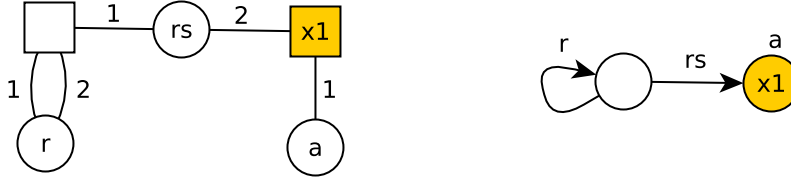


Fig. 3. Graph representations of the query $\exists z_1 \exists z_2 (x_1 = z_2 \wedge rz_1 z_1 \wedge rz_1 z_2 \wedge sz_1 z_2 \wedge az_2)$.

Conjunctive queries can be drawn as graphs. We assume a representation as in (8). Fig. 3 shows two graphs for the query $\exists z_1 \exists z_2 (x_1 = z_2 \wedge rz_1 z_1 \wedge rz_1 z_2 \wedge sz_1 z_2 \wedge az_2)$. The left graph is drawn similar to a conceptual graph. Rectangular nodes are called *object nodes* and represent bound variables. Rounded nodes are called *relation nodes*, and are connected to object nodes by edges labeled $1, \dots, n$, where n is the *arity* of the relation node. A relation node is labeled with one or more relation symbols, and a label m represents an atom $mz_{i_1} \dots z_{i_n}$, where z_{i_k} is the object node connected by the k -edge for $1 \leq k \leq n$. An atom $x_i = z_j$, where x_i is free and z_j is bound, is represented by a label x_i on the object node for z_j . Two relation nodes of the same arity n connected to the same object nodes in the same order $1, \dots, n$ is not permitted. Instead, the respective labels must be concentrated on a single relation node (a convention like this ensures that graphs support an intuitive notion of homomorphisms, formally introduced in Section 3.3). If all relation symbols have arity $n \geq 2$, we can represent conjunctive queries using the standard notation for directed graphs (cf. the right graph in Fig. 3). Object nodes are then drawn as circles, a label m on an object node z_i represents an atom mz_i , and a label m on a directed edge from z_{i_1} to z_{i_2} represents an atom $mz_{i_1} z_{i_2}$. We use the second notation for the examples in this paper, because it takes less space.

3.2 Tableau Queries

Tableau queries are structural representations of conjunctive queries (cf. [1]). We define a tableau query as a pair (T, λ) consisting of a finite, non-empty relational structure T and a partial function $\lambda : \mathbf{var} \rightarrow |T|$ with finite domain of definition. The tableau query for a primitive positive formula φ , given as in (8), is the pair

$(T_\varphi, \lambda_\varphi)$ given by

$$|T_\varphi| := \{z_1, \dots, z_{\ell+k}\}, \quad (9)$$

$$m^{T_\varphi} := \{(y_1, \dots, y_n) \mid my_1 \dots y_n \text{ is an atom in } \varphi\}, \quad (10)$$

$$\lambda_\varphi : \begin{cases} \{x_1, \dots, x_k\} \rightarrow |T_\varphi| \\ x_i \mapsto z_{\ell+i} \end{cases} \quad (11)$$

for all $m \in M_n$ and $n \geq 1$. The partial function λ_φ is called the *summary* of the tableau query (cf. [1, p. 43]).

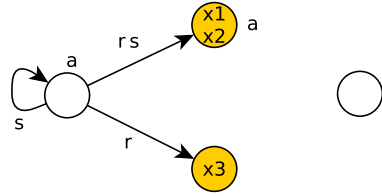


Fig. 4. Conjunctive query (drawn as a graph).

a	r	s
z ₁	z ₁ z ₂	z ₁ z ₁
z ₂	z ₁ z ₃	z ₁ z ₂

x ₁ , x ₂	x ₃
z ₁ z ₂	z ₃ z ₄

Fig. 5. Conjunctive query (drawn as tables).

Figs. 4 and 5 show the query $\exists z_1 \exists z_2 \exists z_3 \exists z_4 (x_1 = z_2 \wedge x_2 = z_2 \wedge x_3 = z_3 \wedge a z_1 \wedge a z_2 \wedge r z_1 z_2 \wedge r z_1 z_3 \wedge s z_1 z_1 \wedge s z_1 z_2)$ as a graph and as tables, respectively. The bottom table in Fig. 5 represents the summary; its second row lists all elements in the carrier set. A representation as tables is more faithful to the definition of a tableau query, but the graph representation seems more comprehensible, so we use it for tableau queries nonetheless.

The following proposition provides a structural characterization for the satisfaction of M -formulas. It is convenient to consider satisfaction also for interpretations (Δ, μ) where μ is only partially defined, i.e. $\mu : \mathbf{var} \rightarrow |\Delta|$. Such a variant can be found e.g. in Abiteboul et al. [1, p. 24f.]. We call such (Δ, μ) a *partial M -interpretation*, and consider a primitive positive formula φ to be false under (Δ, μ) if $\mu(x)$ is undefined for some $x \in \text{free}(\varphi)$ (otherwise, satisfaction is defined as usual). This means that $x=x$ is not a tautology; it is wrong under any interpretation (Δ, μ) that does not define x . We thereby obtain the following proposition:

Proposition 2. *Let $(T_\varphi, \lambda_\varphi)$ be the tableau query for a primitive positive M -formula φ , and let (Δ, μ) be a partial M -interpretation. Then $(\Delta, \mu) \models \varphi$ if and only if there exists a homomorphism $f : T_\varphi \rightarrow \Delta$ with $f \circ \lambda_\varphi \leq \mu$.*

Proof. We consider φ to be given as in (8). By (9) and (10), an assignment $f : \{z_1, \dots, z_{\ell+k}\} \rightarrow |\Delta|$ satisfies $(\Delta, f) \models B_1 \wedge \dots \wedge B_r$ if and only if it is a homomorphism $f : T_\varphi \rightarrow \Delta$. For $(\Delta, f) \models x_1 = z_{\ell+1} \wedge \dots \wedge x_k = z_{\ell+k}$ to

hold in addition, f must be defined on $\{x_1, \dots, x_k\}$ by $f \circ \lambda_\varphi = f|_{\{x_1, \dots, x_k\}}$ (cf. (11)). Finally, an assignment $\mu : \mathbf{var} \rightarrow |T_\varphi|$ satisfies $(\Delta, \mu) \models \varphi$ if and only if $\mu|_{\{x_1, \dots, x_k\}} = f|_{\{x_1, \dots, x_k\}}$ with f as before, i.e. $f \circ \lambda_\varphi \leq \mu$. $\square$

The *result* of a tableau query (T, λ) in Δ is defined as the table

$$\text{res}_\Delta(T, \lambda) := \{f \circ \lambda \mid f : T \rightarrow \Delta\}, \quad (12)$$

and Prop. 2 shows that $\text{res}_\Delta(T_\varphi, \lambda_\varphi)$ is indeed the result of φ in Δ as defined in (7). Note that we have not required $|T|$ to be a set of variables (and it is not required in (12) either). However, replacing the elements of $|T|$ in (T, λ) with distinct variables in $\mathbf{var} \setminus \text{def}(\lambda)$ produces a tableau query $(T_\varphi, \lambda_\varphi)$ with the same result (cf. (8)-(12)). Tableau queries are thus precisely the primitive positive formulas (conjunctive calculus queries).

3.3 Windowed Structures and Homomorphisms

The homomorphism theorem of database theory [1] characterizes query containment by homomorphism [1]. This presupposes a structural representation of conjunctive queries, such as tableau queries. In the original version by Chandra and Merlin [7], queries have been represented by relational structures, equipped with an additional unary relation for each free variable (and also each constant), with homomorphisms as in Section 2. In these works, only queries with the same set of free variables are compared, and query containment then corresponds to logical implication of formulas, as defined in logic textbooks [10].

In this paper, we consider query containment for arbitrary queries (not necessarily in the same free variables), based on the expected definition of homomorphisms (as functions that preserve relations and free variables). An extended version of the homomorphism theorem is stated in Section 3.6, and query containment is related to a table order, introduced in Section 3.5, where the infimum is the well-known natural join operation. A meaningful interpretation of query containment in this context can be given from a logical perspective, based on the notion of p-implication, to be introduced in Section 3.4.

In this section, we define windowed structures as an infinite extension of tableau queries, which are used to model not only queries, but also logical interpretations as defined by Ebbinghaus et al. [10], which allows to formally unify query containment and query solution in Section 3.4.

A *windowed M -structure* is a pair $(\mathfrak{A}, \lambda)$ consisting of an M -structure $\mathfrak{A}$ and a *window* $\lambda : \mathbf{var} \rightarrow |\mathfrak{A}|^1$. A windowed structure $(\mathfrak{A}, \lambda)$ is *finite* if both $\mathfrak{A}$ and $\text{def}(\lambda)$ are finite. The windowed M -structure $(\mathbf{0}_M, \emptyset)$ is called *empty*. Formally, a tableau query is a non-empty finite windowed structure, and a partial M -interpretation is a non-empty windowed M -structure. We use WS_M to denote the class of windowed M -structures, and set $\text{WS}_M[X] := [(\mathfrak{A}, \lambda) \in \text{WS}_M \mid \text{def}(\lambda) = X]$ for $X \subseteq \mathbf{var}$.

¹ We use an alternative definition with the window in the last component to accommodate the existing definitions of tableau queries and M -interpretations.

A *homomorphism* $f : (\mathfrak{A}_1, \lambda_1) \rightarrow (\mathfrak{A}_2, \lambda_2)$ of windowed structures is a homomorphism $f : \mathfrak{A}_1 \rightarrow \mathfrak{A}_2$ with $f \circ \lambda_1 \leq \lambda_2$. We write $(\mathfrak{A}_1, \lambda_1) \lesssim (\mathfrak{A}_2, \lambda_2)$ to indicate that a homomorphism $f : (\mathfrak{A}_1, \lambda_1) \rightarrow (\mathfrak{A}_2, \lambda_2)$ exists. For each $(\mathfrak{A}, \lambda)$, the identity $\text{id}_{|\mathfrak{A}|} : |\mathfrak{A}| \rightarrow |\mathfrak{A}|$ is a homomorphism $\text{id}_{|\mathfrak{A}|} : (\mathfrak{A}, \lambda) \rightarrow (\mathfrak{A}, \lambda)$, and homomorphisms are closed under composition. This means that $\lesssim$ is a reflexive and transitive relation (i.e. a preorder) on WS_M , and we call it the *homomorphism preorder*. The pairs $(\text{WS}_M, \lesssim)$ and $(\text{WS}_M[X], \lesssim)$, with $X \subseteq \mathbf{var}$, are thus preordered classes. Windowed structures $W_1, W_2 \in \text{WS}_M$ are *equivalent*, written $W_1 \simeq W_2$, if $W_1 \lesssim W_2$ and $W_2 \lesssim W_1$. An *isomorphism* is a homomorphism $f : W_1 \rightarrow W_2$ which has an inverse homomorphism $f^{-1} : W_2 \rightarrow W_1$. Such W_1 and W_2 are *isomorphic*, written $W_1 \cong W_2$, which implies $W_1 \simeq W_2$.

The empty windowed structure $(\mathbf{0}_M, \emptyset)$ is the least element of WS_M . The windowed structure $(\mathbf{0}_{M,X}, \text{id}_X)$ with $|\mathbf{0}_{M,X}| = \{X\}$ and $m^{\mathbf{0}_{M,X}} = \emptyset$ for all $m \in M$, is a least element of $\text{WS}_M[X]$. It is a query with $\#X$ isolated nodes, each representing a free variable. The windowed structure $(\mathbf{1}_M, \theta_X)$ with $|\mathbf{1}_M| = \{\emptyset\}$, $m^{\mathbf{1}_M} = \{(\emptyset, \dots, \emptyset)\}$ for all $m \in M$, and $\theta_X(x) = \emptyset$ for all $x \in X$, is a greatest element of $\text{WS}_M[X]$. It is a query with a single node, related to itself by all $m \in M$, and with $\#X$ free variables, all understood to be equal. The windowed structure $(\mathbf{1}_M, \theta_{\mathbf{var}})$ is a greatest element of WS_M .

3.4 Homomorphism Theorem

We now consider the equivalence

$$(\mathfrak{A}_1, \lambda_1) \lesssim (\mathfrak{A}_2, \lambda_2) \Leftrightarrow \begin{array}{l} \text{if } (\mathfrak{A}_2, \lambda_2) \lesssim (\Delta, \mu) \text{ then } (\mathfrak{A}_1, \lambda_1) \lesssim (\Delta, \mu) \\ \text{for all } (\Delta, \mu) \text{ with property } P \end{array} \quad (13)$$

for different properties P of windowed M -structures. The left-to-right implication holds by transitivity regardless of P , and the right-to-left implication holds by reflexivity if $(\mathfrak{A}_2, \lambda_2)$ has property P . Note that the right-hand side of (13) corresponds to the model-theoretic definition of logical implication if (Δ, μ) is an M -interpretation, $(\mathfrak{A}_1, \lambda_1)$ and $(\mathfrak{A}_2, \lambda_2)$ are tableau queries (i.e. non-empty and finite) and $\lesssim$ is read as satisfaction (cf. Prop. 2). To make the right-to-left implication hold in this case, we regard the (Δ, μ) as partial interpretations (cf. Section 3.2), i.e. P is the property that Δ is non-empty. Accordingly, given primitive positive formulas φ and ψ , we say that φ *p-implies* ψ , written $\varphi \models_p \psi$, if every partial interpretation that satisfies φ also satisfies ψ . Equation (13) thus characterizes homomorphisms by p-implication, and we summarize this in the following variant of the homomorphism theorem of database theory.

Theorem 1 (Homomorphism Theorem (Logic Version)). *Let φ and ψ be primitive positive formulas. Then $(T_\psi, \lambda_\psi) \lesssim (T_\varphi, \lambda_\varphi)$ if and only if φ p-implies ψ .*

We now compare p-implication to the usual notion of implication, as defined in Section 2. While extending satisfaction to partial interpretations as in Section 3.2 may seem unspectacular, we obtain as a consequence that an equality

$x = x$ is no longer a tautology, because it does not hold under interpretations where x is undefined. Likewise, $x = x$ does not imply $y = y$, because there exist interpretations where x is defined and y is undefined. Proposition 3 characterizes p-implication.

Proposition 3. *Let φ and ψ be primitive positive formulas. Then*

$$\varphi \models_p \psi \Leftrightarrow \varphi \models \psi \text{ and } \text{free}(\psi) \subseteq \text{free}(\varphi). \quad (14)$$

The notions of implication and p-implication coincide if $\text{free}(\varphi) = \text{free}(\psi)$.

Proof. If $\varphi \models_p \psi$ holds, then $\varphi \models \psi$ follows a fortiori and $\text{free}(\psi) \subseteq \text{free}(\varphi)$ by contraposition. Now assume the right-hand side of (14) holds, and let (Δ, μ) be a partial interpretation with $(\Delta, \mu) \models \varphi$. Then $(\Delta, \tilde{\mu}) \models \varphi$ for any total $\tilde{\mu} \geq \mu$, $(\Delta, \tilde{\mu}) \models \psi$ follows from $\varphi \models \psi$, and finally $(\Delta, \mu) \models \psi$ because of $\text{free}(\psi) \subseteq \text{free}(\varphi) \subseteq \text{def}(\mu)$. This shows $\varphi \models_p \psi$. $\square$

In Chandra and Merlin's paper [7], a database was considered to be finite. For tableau queries, this does not lead to a different notion of implication, because (13) also holds if P is finiteness.

On the basis of Prop. 2, we generalize satisfaction to arbitrary windowed M -structures, i.e. we say that (Δ, μ) *satisfies* $(\mathfrak{A}, \lambda)$ if $(\mathfrak{A}, \lambda) \lesssim (\Delta, \mu)$, where $(\mathfrak{A}, \lambda)$ is considered a query and (Δ, μ) an interpretation. The result of a query is then given by the function

$$\text{res}_\Delta : \begin{cases} \text{WS}_M \rightarrow \text{Tab}_G \\ (\mathfrak{A}, \lambda) \mapsto \{\mu|_{\text{def}(\lambda)} \mid (\mathfrak{A}, \lambda) \lesssim (\Delta, \mu)\} \end{cases}, \quad (15)$$

in accordance with (12). We have thus generalized conjunctive queries to infinite queries. The *empty query* $(\mathbf{0}_M, \emptyset)$ states nothing and is always true, whereas the *empty interpretation* $(\mathbf{0}_M, \emptyset)$ describes an empty universe in which only the empty query holds.

If we consider (13) for arbitrary windowed structures (and P as a tautology), the right-hand side says that $(\mathfrak{A}_2, \lambda_2)$ implies $(\mathfrak{A}_1, \lambda_1)$ in the model-theoretic sense. Because partial interpretations are considered, we write this as a p-implication $(\mathfrak{A}_2, \lambda_2) \models_p (\mathfrak{A}_1, \lambda_1)$. Then (13) states that

$$(\mathfrak{A}_1, \lambda_1) \lesssim (\mathfrak{A}_2, \lambda_2) \Leftrightarrow (\mathfrak{A}_2, \lambda_2) \models_p (\mathfrak{A}_1, \lambda_1), \quad (16)$$

so $(\text{WS}_M, \models_p)$ is a preordered class, and it is dual to $(\text{WS}_M, \lesssim)$. We have obtained that the $\lesssim$ -relation on WS_M describes both satisfaction and p-implication, and whether it is read as one or the other depends on the context; on the formal level, the two notions coincide.

A logic-based version of the result operation in (15), where the result is the set of partial assignments that satisfy the query, is given by

$$\text{res}_\Delta^* : \begin{cases} \text{WS}_M \rightarrow \mathfrak{P}(\text{Tup}(G)) \\ (\mathfrak{A}, \lambda) \mapsto \{\mu \in \text{Tup}(G) \mid (\mathfrak{A}, \lambda) \lesssim (\Delta, \mu)\} \end{cases}. \quad (17)$$

It is obtained from (13) that

$$W_1 \lesssim W_2 \Leftrightarrow \begin{array}{l} \text{res}_\Delta^*(W_2) \subseteq \text{res}_\Delta^*(W_1) \\ \text{for all relational structures } \Delta \end{array} \quad (18)$$

holds for all $W_1, W_2 \in \text{WS}_M$. Note that we have allowed interpretations in an empty universe, so empty databases can be represented by $\Delta = \mathbf{0}_M$, and in this case $\text{res}_{\mathbf{0}_M}^*(W_1) = \text{res}_{\mathbf{0}_M}^*(W_2) = \emptyset$.

To obtain an analog of (18) for the result operation in (15), we need a means to compare result tables. A table order will be introduced in Section 3.5, and the homomorphism theorem is restated for tables in Section 3.6. Of course, when we restrict (18) to queries $W_1, W_2 \in \text{WS}_M[X]$, tables $\text{res}_\Delta(W_1)$ and $\text{res}_\Delta(W_2)$ are simply compared as subsets of G^X . A corresponding statement of the homomorphism theorem can be found in Abiteboul et al. [1, p. 115].

3.5 Tables

A table can be seen as a minimal representation of a set of partial assignments, where the full set of assignments is provided by the function

$$\text{pset} : \begin{cases} \text{Tab}_G \rightarrow \mathfrak{P}(\text{Tup}(G)) \\ T \mapsto \{\mu \in \text{Tup}(G) \mid \mu|_{\text{schema}(T)} \in T\} \end{cases} \quad (19)$$

Every table $T \in \text{Tab}_G$ can be reconstructed from $\text{pset}(T)$ by the function

$$\text{table} : \begin{cases} \mathfrak{P}(\text{Tup}(G)) \rightarrow \text{Tab}_G \\ A \mapsto \{\mu|_{\text{schema}(A)} \mid \mu \in A\} \end{cases}, \quad (20)$$

where $\text{schema}(A) := \bigcap_{\mu \in A} \text{def}(\mu)$. The functions pset and table thus establish a one-to-one correspondence between Tab_G and $\text{pset}(\text{Tab}_G)$.

The following Prop. 4 shows that the *table order*, defined by

$$T_1 \leq T_2 :\Leftrightarrow \begin{array}{l} \text{schema}(T_2) \subseteq \text{schema}(T_1) \\ \text{and } \{\lambda|_{\text{schema}(T_2)} \mid \lambda \in T_1\} \subseteq T_2 \end{array} \quad (21)$$

for tables $T_1, T_2 \in \text{Tab}_G$, reflects the set order on $\mathfrak{P}(\text{Tup}(G))$. The right-hand side in (21) means that the first table, projected to the column set of the second table, must be a subset of the second table.

Proposition 4. *The function $\text{pset} : (\text{Tab}_G, \leq) \rightarrow (\mathfrak{P}(\text{Tup}(G)), \subseteq)$ is an order embedding, i.e.*

$$T_1 \leq T_2 \Leftrightarrow \text{pset}(T_1) \subseteq \text{pset}(T_2) \quad (22)$$

for all $T_1, T_2 \in \text{Tab}_G$.

Proof. For the left-to-right implication in (22), we assume $T_1 \leq T_2$ and $t \in \text{pset}(T_1)$. Then $t|_{\text{schema}(T_1)} \in T_1$ by (19), $t|_{\text{schema}(T_2)} \in T_2$ by the assumption, and $t \in \text{pset}(T_2)$ by (19). This shows $\text{pset}(T_1) \subseteq \text{pset}(T_2)$. The right-to-left implication is shown in the same way. $\square$

The *natural join* of a family $(T_i)_{i \in I}$ of tables is the table

$$\bigbowtie_{i \in I} T_i := \{\lambda \in G^{\bigcup_{i \in I} \text{schema}(T_i)} \mid \lambda|_{\text{schema}(T_i)} \in T_i \text{ for all } i \in I\}, \quad (23)$$

cf. [1, p.58]. The join of the empty family is the table $\{\emptyset\}$ with schema $\emptyset$ and a single row. This is the largest table in the table order. In connection with yes-no-queries, it is interpreted as the yes-answer (cf. [1, p.42]). It is straightforward to verify that the natural join is the infimum operation in $(\text{Tab}_G, \leq)$. In particular, every subset of Tab_G has an infimum, so $(\text{Tab}_G, \leq)$ is a complete lattice (cf. [16, Prop. 1]). The pset function preserves infima, i.e.

$$\text{pset}(\bigbowtie_{i \in I} T_i) = \bigcap_{i \in I} \text{pset}(T_i) \quad (24)$$

for all families $(T_i)_{i \in I}$ in Tab_G . The join of tables thus corresponds to the intersection of their representations in $\mathfrak{P}(\text{Tup}(G))$.

The supremum in the table order has been called the *co-join* in [21], and it is given by

$$\bigbowtie_{i \in I} T_i := \bigcup_{i \in I} \{\lambda|_{\bigcap_{i \in I} \text{schema}(T_i)} \mid \lambda \in T_i\}. \quad (25)$$

The co-join of the empty family is the empty table. It is the smallest table in the table order.

Fig. 6 shows an example of the join and co-join operations. For tables with the same schema, join and co-join simply correspond to the intersection and union of rows, respectively.

x_1	x_2	x_3	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_2	x_3
1	2	3	2	3	4	1	2	3	4	2	3
6	7	8	2	3	5	1	2	3	5	7	8
T_1			T_2			$T_1 \bowtie T_2$				$T_1 \bowtie T_2$	

Fig. 6. Join and co-join

3.6 Query Containment and Equivalence

Given queries W_1 and W_2 , we say that W_1 *contains* W_2 if $\text{res}_\Delta(W_2) \leq \text{res}_\Delta(W_1)$ for all relational structures Δ .

Theorem 2 (Homomorphism Theorem (Database Theory Version)).

Let $W_1, W_2 \in \text{WS}_M$. Then $W_1 \lesssim W_2$ if and only if W_1 contains W_2 , i.e.

$$W_1 \lesssim W_2 \Leftrightarrow \begin{array}{l} \text{res}_\Delta(W_2) \leq \text{res}_\Delta(W_1) \\ \text{for all relational structures } \Delta \end{array} . \quad (26)$$

Proof. The following equivalences

$$W_1 \lesssim W_2 \Leftrightarrow \forall \Delta : \text{res}_\Delta^*(W_2) \subseteq \text{res}_\Delta^*(W_1) \quad (27)$$

$$\Leftrightarrow \forall \Delta : \text{pset}(\text{res}_\Delta(W_2)) \subseteq \text{pset}(\text{res}_\Delta(W_1)) \quad (28)$$

$$\Leftrightarrow \forall \Delta : \text{res}_\Delta(W_2) \leq \text{res}_\Delta(W_1) \quad (29)$$

follow from (18), $\text{res}_\Delta^* = \text{pset} \circ \text{res}_\Delta$ and Prop. 4, respectively. $\square$

Abiteboul et al. [1] define query containment for tableau queries as above, but consider only finite databases. As remarked in Section 3.2, the restriction to finite Δ leads to the same notion of implication for tableau queries, so if we require W_1 and W_2 to be tableau queries, we can require on the right-hand side of (18) and (26) that Δ is finite.

Queries that contain each other are called *equivalent*. By Thm. 2, query equivalence coincides with hom-equivalence, and we shall use one to mean the other.

3.7 Query Minimization

Theorem 3 (Query Minimization). *Let (T, λ) be a tableau query. There exists a minimal tableau query (S, ν) with $(S, \nu) \simeq (T, \lambda)$, and it is unique up to isomorphism. There exists an idempotent endomorphism $h : (T, \lambda) \rightarrow (T, \lambda)$ with $(T|_{\text{rng}(h)}, \lambda) \cong (S, \nu)$.*

Proof. The existence of a minimal (S, ν) with $(S, \nu) \simeq (T, \lambda)$ follows from the well-ordering of natural numbers. For any other such query (S', ν') , there exist $f : (S, \nu) \rightarrow (S', \nu')$ and $g : (S', \nu') \rightarrow (S, \nu)$, and $g \circ f : (S, \nu) \rightarrow (S, \nu)$ is bijective, because $(S, \nu) \simeq (S|_{\text{rng}(g \circ f)}, \nu)$ and (S, ν) is minimal. Any bijection $k : (S, \nu) \rightarrow (S, \nu)$ acts as a permutation on the (finite!) relations m^S , i.e. $k(m^S) = m^S$, which means that k is an isomorphism (where we have defined $k(m^S) := \{(k(a_1), \dots, k(a_n)) \mid (a_1, \dots, a_n) \in m^S\}$ for $m \in M$ and $n \geq 1$). Thus $(g \circ f)^{-1}$, and by symmetry $(f \circ g)^{-1}$, are homomorphisms and provide left and right inverses $k := (g \circ f)^{-1} \circ g$ and $k' := g \circ (f \circ g)^{-1}$ of f . Moreover, we have $k = k \circ f \circ k' = k'$, so f is an isomorphism. This shows that (S, ν) is unique up to isomorphism.

Let $s : (S, \nu) \rightarrow (T, \lambda)$ be arbitrary. As before, we obtain a left inverse s_ℓ , so that $s_\ell \circ s = \text{id}_{|S|}$. Then $\text{rng}(s_\ell) = |S|$, and thus $\text{rng}(h) = \text{rng}(s)$ for $h := s \circ s_\ell$. Then $(S, \nu) \simeq (T|_{\text{rng}(h)}, \lambda)$ and $(T|_{\text{rng}(h)}, \lambda)$ is minimal, so $(S, \nu) \cong (T|_{\text{rng}(h)}, \lambda)$ by uniqueness of the minimal query. Finally, $h : (T, \lambda) \rightarrow (T, \lambda)$ is idempotent because $h \circ h = s \circ (s_\ell \circ s) \circ s_\ell = h$. $\square$

Example Let M be the signature $M_1 \cup M_2$ with $M_1 = \{a, b\}$ and $M_2 = \{r\}$. The upper graph in Fig. 7 represents a windowed structure $(\mathcal{G}_1, \lambda_1) \in \text{WS}_M[\{x\}]$ with $|\mathcal{G}_1| = \{1, 2, 3, 4\}$, $a^{\mathcal{G}_1} = \{1, 4\}$, $b^{\mathcal{G}_1} = \{3, 4\}$, $r^{\mathcal{G}_1} = \{(1, 2), (3, 2), (4, 2)\}$ and $\lambda_1 = \{(x, 1)\}$. Each $i \in |\mathcal{G}_1|$ is written inside its node, so we can refer to it (we

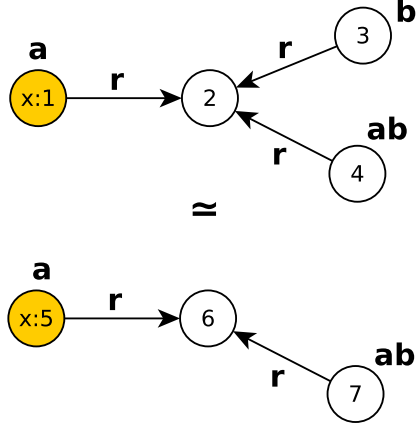


Fig. 7. Equivalence and minimization.

write $x:i$ to separate it from the window variable x). The lower graph represents a windowed structure $(\mathcal{G}_2, \lambda_2) \in \text{WS}_M[\{x\}]$, its definition can be obtained from the graph. The function $f : (\mathcal{G}_1, \lambda_1) \rightarrow (\mathcal{G}_2, \lambda_2)$ with $f(1) = 5$, $f(2) = 6$ and $f(3) = f(4) = 7$ preserves all relations, which shows $(\mathcal{G}_1, \lambda_1) \lesssim (\mathcal{G}_2, \lambda_2)$. The function $g : (\mathcal{G}_2, \lambda_2) \rightarrow (\mathcal{G}_1, \lambda_1)$ with $g(5) = 1$, $g(6) = 2$ and $g(7) = 4$ also preserves all relations, which shows $(\mathcal{G}_2, \lambda_2) \lesssim (\mathcal{G}_1, \lambda_1)$. This shows $(\mathcal{G}_1, \lambda_1) \simeq (\mathcal{G}_2, \lambda_2)$. Moreover, g identifies $(\mathcal{G}_2, \lambda_2)$ with a subgraph of $(\mathcal{G}_1, \lambda_1)$, and f is a folding onto that subgraph. Since $(\mathcal{G}_2, \lambda_2)$ can not be folded onto a proper subgraph of itself, it is minimal.

4 Conjunctive Queries: Sum and Product

In Section 3, we formalized conjunctive queries, their infinite generalizations, and query containment, by windowed structures and the preorder of homomorphisms (cf. Section 3.6). The sum and product of queries, to be introduced in Sections 4.2 and 4.1, are supremum and infimum operations (modulo equivalence) in that preorder. The sum of queries corresponds to their logical conjunction. The product construction has been previously used by Baader et al. [4] in a similar context (the product does not correspond to disjunction, which is not available for conjunctive queries). In Section 4.3, we eliminate query equivalence, thereby obtaining a query order with the properties of a complete lattice. A technical problem faced via this process is that queries form a proper class, rather than a set. Dealing with a proper class can be avoided by considering finite queries only (cf. Section 6.1), but this is at the expense of the completeness property.

4.1 Product of Windowed Structures

Definition 1 (Product of windowed M -structures). Let $((\mathfrak{A}_i, \lambda_i))_{i \in I}$ be a family of windowed M -structures. The product $\prod_{i \in I} (\mathfrak{A}_i, \lambda_i)$ is defined as the windowed M -structure $(\mathfrak{A}_\Pi, \lambda_\Pi)$ with

$$|\mathfrak{A}_\Pi| := \bigtimes_{i \in I} |\mathfrak{A}_i|, \quad (30)$$

$$((a_i^{(1)})_{i \in I}, \dots, (a_i^{(n)})_{i \in I}) \in m^{\mathfrak{A}_\Pi} \Leftrightarrow (a_i^{(1)}, \dots, a_i^{(n)}) \in m^{\mathfrak{A}_i} \text{ f.a. } i \in I, \quad (31)$$

$$\lambda_\Pi : \begin{cases} \bigcap_{i \in I} \text{def}(\lambda_i) \rightarrow |\mathfrak{A}_\Pi| \\ x \mapsto (\lambda_i(x))_{i \in I} \end{cases} \quad (32)$$

for all $m \in M_n$ and $n \geq 1$.

Proposition 5. The product $\prod_{i \in I} (\mathfrak{A}_i, \lambda_i)$ is a greatest lower bound of $\{(\mathfrak{A}_i, \lambda_i) \mid i \in I\}$ in $(\text{WS}_M, \lesssim)$.

Proof. Let $(\mathfrak{A}_\Pi, \lambda_\Pi) := \prod_{i \in I} (\mathfrak{A}_i, \lambda_i)$. We first show that the k -th projection

$$\pi_k : \begin{cases} |\mathfrak{A}_\Pi| \rightarrow |\mathfrak{A}_k| \\ ((a_i)_{i \in I}) \mapsto a_k \end{cases} \quad (33)$$

is a homomorphism $\pi_k : (\mathfrak{A}_\Pi, \lambda_\Pi) \rightarrow (\mathfrak{A}_k, \lambda_k)$, which means that $(\mathfrak{A}_\Pi, \lambda_\Pi)$ is a lower bound of $(\mathfrak{A}_k, \lambda_k)$ for all $k \in I$. Given $((a_i^{(1)})_{i \in I}, \dots, (a_i^{(n)})_{i \in I}) \in m^{\mathfrak{A}_\Pi}$, we have $(\pi_k((a_i^{(1)})_{i \in I}), \dots, \pi_k((a_i^{(n)})_{i \in I})) = (a_k^{(1)}, \dots, a_k^{(n)}) \in m^{\mathfrak{A}_k}$ by (31), so π_k preserves relations. Using (32), we obtain $(\pi_k \circ \lambda_\Pi)(x) = \pi_k((\lambda_i(x))_{i \in I}) = \lambda_k(x)$ for $x \in \text{def}(\lambda_\Pi)$, so π_k preserves the window.

Now let $(\mathfrak{A}, \lambda)$ be a lower bound of $\{(\mathfrak{A}_i, \lambda_i) \mid i \in I\}$, which means that homomorphisms $f_i : (\mathfrak{A}, \lambda) \rightarrow (\mathfrak{A}_i, \lambda_i)$ exist for all $i \in I$. We then show that

$$f : \begin{cases} |\mathfrak{A}| \rightarrow |\mathfrak{A}_\Pi| \\ a \mapsto (f_i(a))_{i \in I} \end{cases} \quad (34)$$

is a homomorphism $f : (\mathfrak{A}, \lambda) \rightarrow (\mathfrak{A}_\Pi, \lambda_\Pi)$, and thus that $(\mathfrak{A}_\Pi, \lambda_\Pi)$ is a greatest lower bound. So let $(a_1, \dots, a_n) \in m^{\mathfrak{A}}$ for some $m \in M_n$ and $n \geq 1$. Then $(f_i(a_1), \dots, f_i(a_n)) \in m^{\mathfrak{A}_i}$ for all $i \in I$, and by (31) we have $(f(a_1), \dots, f(a_n)) \in m^{\mathfrak{A}_\Pi}$. It is easily verified that $f \circ \lambda \leq \lambda_\Pi$, so f preserves the window. $\square$

Fig. 8 shows the product of two windowed structures. A product node (i, j) shows what the nodes i and j have in common. For example, $(2, 6)$ has an attribute b and an r -successor, and that is precisely what 2 and 5 have in common.

4.2 Sum of Windowed Structures

Definition 2 (Sum of windowed M -structures). Let $((\mathfrak{A}_i, \lambda_i))_{i \in I}$ be a family of windowed M -structures. Let σ be the smallest equivalence relation on the disjoint union $\bigsqcup_{i \in I} |\mathfrak{A}_i|$ such that

$$(i, \lambda_i(x)) \sigma (j, \lambda_j(x)) \quad (35)$$

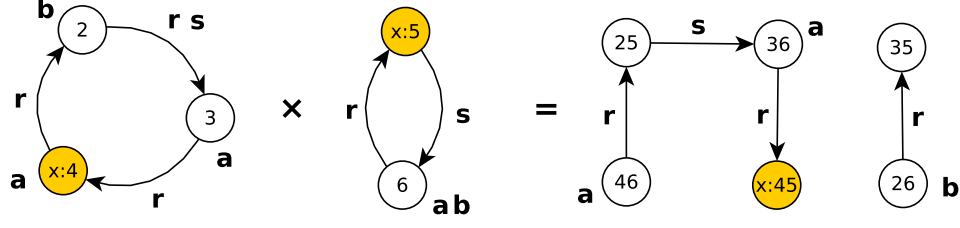


Fig. 8. Product of windowed structures.

for all $i, j \in I$ and $x \in \text{def}(\lambda_i) \cap \text{def}(\lambda_j)$. The sum $\sum_{i \in I} (\mathfrak{A}_i, \lambda_i)$ is defined as the windowed M -structure $(\mathfrak{A}_\Sigma, \lambda_\Sigma)$ with

$$|\mathfrak{A}_\Sigma| := \{[(i, a)]_\sigma \mid (i, a) \in \bigsqcup_{i \in I} |\mathfrak{A}_i|\}, \quad (36)$$

$$m^{\mathfrak{A}_\Sigma} := \bigcup_{i \in I} \{([(i, a_1)]_\sigma, \dots, [(i, a_n)]_\sigma) \mid (a_1, \dots, a_n) \in m^{\mathfrak{A}_i}\}, \quad (37)$$

$$\lambda_\Sigma : \begin{cases} \bigcup_{i \in I} \text{def}(\lambda_i) \rightarrow |\mathfrak{A}_\Sigma| \\ x \mapsto [(j, \lambda_j(x))]_\sigma \text{ (where } j \in I \text{ with } x \in \text{def}(\lambda_j) \text{ is arbitrary)} \end{cases} \quad (38)$$

for all $m \in M$ and $n \geq 1$.

Note that λ_Σ in (38) is well-defined because of (35).

Proposition 6. The sum $\sum_{i \in I} (\mathfrak{A}_i, \lambda_i)$ is a least upper bound of $\{(\mathfrak{A}_i, \lambda_i) \mid i \in I\}$ in $(\text{WS}_M, \lesssim)$.

Proof. Let $(\mathfrak{A}_\Sigma, \lambda_\Sigma) := \sum_{i \in I} (\mathfrak{A}_i, \lambda_i)$. We first show that each function

$$\iota_k : \begin{cases} |\mathfrak{A}_k| \rightarrow |\mathfrak{A}_\Sigma| \\ a \mapsto [(k, a)]_\sigma \end{cases} \quad (39)$$

is a homomorphism $\iota_k : (\mathfrak{A}_k, \lambda_k) \rightarrow (\mathfrak{A}_\Sigma, \lambda_\Sigma)$, which means that $(\mathfrak{A}_\Sigma, \lambda_\Sigma)$ is an upper bound of $(\mathfrak{A}_k, \lambda_k)$ for all $k \in I$. Given $(a_1, \dots, a_n) \in m^{\mathfrak{A}_k}$, we have $(\iota_k(a_1), \dots, \iota_k(a_n)) = ([(k, a_1)], \dots, [(k, a_n)]) \in m^{\mathfrak{A}_\Sigma}$ by (37), so ι_k preserves relations. Using (38), we obtain $(\iota_k \circ \lambda_k)(x) = [(k, \lambda_k(x))] = \lambda_\Sigma(x)$ for $x \in \text{def}(\lambda_k)$, so ι_k preserves the window.

Now let $(\mathfrak{A}, \lambda)$ be an upper bound of $(\mathfrak{A}_i, \lambda_i) \mid i \in I$, which means that homomorphisms $f : (\mathfrak{A}_i, \lambda_i) \rightarrow (\mathfrak{A}, \lambda)$ exist for all $i \in I$. We then show that

$$f : \begin{cases} |\mathfrak{A}_\Sigma| \rightarrow |\mathfrak{A}| \\ [(i, a)] \mapsto f_i(a) \end{cases} \quad (40)$$

is well-defined and $f : (\mathfrak{A}_\Sigma, \lambda_\Sigma) \rightarrow (\mathfrak{A}, \lambda)$ is a homomorphism, and thus that $(\mathfrak{A}_\Sigma, \lambda_\Sigma)$ is a least upper bound.

Let ϱ be the equivalence relation defined by $(i, a) \varrho (j, b) :\Leftrightarrow f_i(a) = f_j(b)$ for all $(i, a), (j, b) \in \bigsqcup_{i \in I} |\mathfrak{A}_i|$. Because of $f_i \circ \lambda_i \leq \lambda$ and $f_j \circ \lambda_j \leq \lambda$, we have $(i, \lambda_i(x)) \varrho (j, \lambda_j(x))$ for all $i, j \in I$ and $x \in \text{def}(\lambda_i) \cap \text{def}(\lambda_j)$. But σ was the smallest equivalence relation with this property (cf. (35)), which implies $\sigma \subseteq \varrho$. We have defined $f([(i, a)]_\sigma) = f_i(a)$ and $f([(j, b)]_\sigma) = f_j(b)$, and for $(i, a) \sigma (j, b)$ we thus have indeed $f_i(a) = f_j(b)$, so f is well-defined. Finally, let $(b_1, \dots, b_n) \in m^{\mathfrak{A}_\Sigma}$ for some $m \in M_n$ and $n \geq 1$. By (37), we have $(b_1, \dots, b_n) = ([(i, a_1)], \dots, [(i, a_n)]) for some $(a_1, \dots, a_n) \in m^{\mathfrak{A}_i}$ and $i \in I$. Then $(f(b_1), \dots, f(b_n)) = (f_i(a_1), \dots, f_i(a_n)) \in m^{\mathfrak{A}}$. It is easily verified that $f \circ \lambda_\Sigma \leq \lambda$, so we obtain $f : (\mathfrak{A}_\Sigma, \lambda_\Sigma) \rightarrow (\mathfrak{A}, \lambda)$. $\square$$

Fig. 9 shows an example of the sum of windowed structures. The graphs are merged by the nodes that correspond to the same query variables, as identified by the window. In Fig. 9, these are the nodes that correspond to x and y . The sum $(T_\varphi, \lambda_\varphi) + (T_\psi, \lambda_\psi)$ corresponds to the conjunction of formulas $\varphi \wedge \psi$.

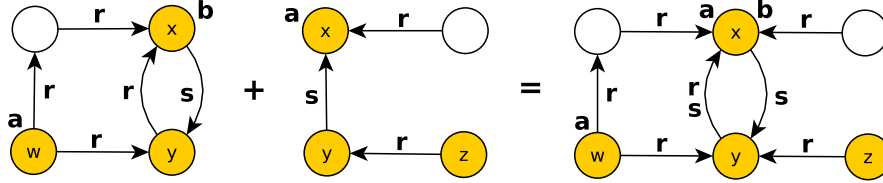


Fig. 9. Sum of windowed structures.

4.3 Complete Lattice Operations on WS_M

Propositions 5 and 6 state that the product and sum of windowed structures are greatest lower and least upper bounds, respectively. But these bounds are unique only up to query equivalence. In this section, we eliminate query equivalence, thus turning the preorder of queries into an order with infima and suprema.

The standard construction that achieves this for preordered sets would be the set of equivalence classes with the induced order. Note however that $(\text{WS}_M, \lesssim)$ is a preordered class, not a preordered set, which presents a technical problem. For a practically oriented reader, it may be sufficient to work only with tableau queries (i.e. finite windowed structures), which are easily captured in a set, but one should be aware that completeness is then lost, because tableau queries are only closed under *finite* suprema and infima. We choose to address the technical problem by defining the query order on a system of representatives, rather than a collection of equivalence classes.

The axiom of choice states that a system of representatives exists for every equivalence relation on a set. But it does not apply here, because WS_M is a proper class. The existence is addressed by the following proposition.

Proposition 7. *There exists a function rep on WS_M with*

$$\text{rep}(W) \simeq W, \quad (41)$$

$$W_1 \simeq W_2 \rightarrow \text{rep}(W_1) = \text{rep}(W_2) \quad (42)$$

for all $W, W_1, W_2 \in \text{WS}_M$.

Proof. For every cardinal α , let $S_{M,\alpha}$ denote the set of all windowed M -structures with carrier set α (the cardinal α is a set with cardinality α , cf. [9]). Let $\text{cls}_\alpha(W) := \{U \in S_{M,\alpha} \mid U \simeq W\}$. Finally, let $\beta(W)$ denote the smallest cardinal α such that $\text{cls}_\alpha(W) \neq \emptyset$, and let $\text{rep}(W) := \sum \text{cls}_{\beta(W)}(W)$.

By Prop. 6, the sum $\sum \text{cls}_{\beta(W)}(W)$ is a least upper bound of $\text{cls}_{\beta(W)}(W)$. Because all elements of $\text{cls}_{\beta(W)}(W)$ are equivalent, $\sum \text{cls}_{\beta(W)}(W)$ is equivalent to each $U \in \text{cls}_{\beta(W)}(W)$, and thus to W , which shows (41). For $W_1 \simeq W_2$, we have $\beta(W_1) = \beta(W_2)$, moreover $\text{cls}_{\beta(W_1)}(W_1) = \text{cls}_{\beta(W_2)}(W_2)$ and thus $\text{rep}(W_1) = \text{rep}(W_2)$, which shows (42). $\square$

For the rest of the paper, it is assumed that rep is always the same fixed function, and besides (41) and (42), its definition will not matter. The proper class

$$\text{WS}_M^{\text{rep}} := [\text{rep}(W) \mid W \in \text{WS}_M] \quad (43)$$

is a system of representatives, and $\text{rep}(W)$ is the representative of W . The restriction of $\lesssim$ to WS_M^{rep} is an order, and we simply use $\leq$ to denote it. So $(\text{WS}_M^{\text{rep}}, \leq)$ is an ordered class, and every subset $\{W_i \mid i \in I\}$ of WS_M^{rep} has an infimum and a supremum, given by

$$\bigwedge_{i \in I} W_i = \text{rep}\left(\prod_{i \in I} W_i\right), \quad (44)$$

$$\bigvee_{i \in I} W_i = \text{rep}\left(\sum_{i \in I} W_i\right), \quad (45)$$

respectively (cf. Props. 5 and 6).

The formal representative $\text{rep}(W)$ should be regarded as an equivalence class of queries rather than an actual query. When queries are drawn, or used in practical computations, any other equivalent graph can be used in place of the formal representative. This is made precise by the following proposition.

Proposition 8. *The following holds for all $W_1, W_2 \in \text{WS}_M$ and all families $(W_i)_{i \in I}$ in WS_M :*

$$W_1 \lesssim W_2 \Leftrightarrow \text{rep}(W_1) \leq \text{rep}(W_2), \quad (46)$$

$$\prod_{i \in I} W_i \simeq \bigwedge_{i \in I} \text{rep}(W_i), \quad (47)$$

$$\sum_{i \in I} W_i \simeq \bigvee_{i \in I} \text{rep}(W_i). \quad (48)$$

Proof. We have $W_1 \lesssim W_2 \Leftrightarrow \text{rep}(W_1) \lesssim \text{rep}(W_2)$ by (41), and $\text{rep}(W_1) \lesssim \text{rep}(W_2) \Leftrightarrow \text{rep}(W_1) \leq \text{rep}(W_2)$ by definition of $\leq$, which shows (46). For (47), consider $\prod_{i \in I} W_i \simeq \prod_{i \in I} \text{rep}(W_i) \simeq \text{rep}(\prod_{i \in I} \text{rep}(W_i)) = \bigwedge_{i \in I} \text{rep}(W_i)$, which follows from Prop. 5, (41) and (44). Equation (48) is shown in the same way. $\square$

5 Concepts of a relational structure

In Section 5.1, we present dual adjoints for the result functions on conjunctive queries, in both the logic-based and database-theoretic variants. The dual adjoints provide the available information on a given set of tuples. Dual adjoints establish a Galois connection; the well-known results are gathered in Section 5.2. As is usual, we define concepts as stable pairs under a Galois connection (cf. [16] and [15]). Concepts are defined in Section 5.3 for the database-theoretic Galois connection (these connect the query order with the table order), and in Section 5.7 for the logic-based Galois connection (which uses sets instead of tables). Both definitions are shown to be equivalent in Section 5.7, i.e. the respective concept lattices are isomorphic. An example concept lattice is presented in Section 5.4 that illustrates this isomorphism. In Section 5.5, we provide insight into the nature of concept intents, and in so doing obtain a descriptive algorithm for computing the concept lattice. In the example concept lattice, intents are drawn as connected graphs, although formally intents comprise further components, unrelated to the query subject. Section 5.6 provides the formal grounds through which this excessive information can safely be ignored.

5.1 Dual Adjoints

The result functions res_Δ and res_Δ^* are antitone. Consider more generally an antitone function $g : (Q, \leq) \rightarrow (P, \leq)$ between ordered classes. If the function

$$f : \begin{cases} (P, \leq) \rightarrow (Q, \leq) \\ p \mapsto \max[y \in Q \mid g(y) \geq p] \end{cases} \quad (49)$$

is well-defined (i.e. if the maximum in eqn. (49) exists), it is called the *dual adjoint* of g . Not every antitone function has a dual adjoint. However, we can establish in this section the dual adjoints of res_Δ and res_Δ^* .

To understand what a dual adjoint is, it is instructive to apply the definition in eqn. (49) to res_Δ^* . In this case, $(\mathfrak{P}(\text{Tup}(G)), \subseteq)$ and $(\text{WS}_M^{\text{rep}}, \leq)$ take the role of $(P, \leq)$ and $(Q, \leq)$, respectively. For a given tuple set $A \subseteq \text{Tup}(G)$, the class $[W \in \text{WS}_M^{\text{rep}} \mid \text{res}_\Delta^*(W) \supseteq A]$ contains all queries which contain A in their result set, and its maximum is the most descriptive query in this class. The dual adjoint thus provides information available in Δ on a given tuple set A .

Proposition 9. *The dual adjoint of res_Δ^* is the function*

$$\text{info}_\Delta^* : \begin{cases} \mathfrak{P}(\text{Tup}(G)) \rightarrow \text{WS}_M^{\text{rep}} \\ A \mapsto \bigwedge_{\mu \in A} \text{rep}(\Delta, \mu) \end{cases} . \quad (50)$$

Proof. According to eqn. (17), a query $W \in \text{WS}_M^{\text{rep}}$ contains $\mu \in A$ in its result set if and only if $W \lesssim (\Delta, \mu)$, or equivalently if $W \leq \text{rep}(\Delta, \mu)$, i.e. if W is a lower bound of $\text{rep}(\Delta, \mu)$. The maximal query in WS_M^{rep} solved by all $\mu \in A$ is thus the infimum $\bigwedge_{\mu \in A} \text{rep}(\Delta, \mu)$. $\square$

Proposition 10. *The dual adjoint of res_Δ is the function*

$$\text{info}_\Delta : \begin{cases} \text{Tab}_G \rightarrow \text{WS}_M^{\text{rep}} \\ T \mapsto \bigwedge_{\mu \in T} \text{rep}(\Delta, \mu) \end{cases} . \quad (51)$$

Proof. Considering eqns. (21), (15) and (17), we obtain

$$\text{res}_\Delta(\mathfrak{A}, \lambda) \geq T \Leftrightarrow \forall \mu \in T : \mu|_{\text{def}(\lambda)} \in \text{res}_\Delta(\mathfrak{A}, \lambda) \quad (52)$$

$$\Leftrightarrow \forall \mu \in T : \mu \in \text{res}_\Delta^*(\mathfrak{A}, \lambda) \quad (53)$$

$$\Leftrightarrow \text{res}_\Delta^*(\mathfrak{A}, \lambda) \supseteq T . \quad (54)$$

The dual adjoint is as in eqn. (50), as $\text{info}_\Delta(T) = \max[W \in \text{WS}_M^{\text{rep}} \mid \text{res}_\Delta(W) \geq T] = \max[W \in \text{WS}_M^{\text{rep}} \mid \text{res}_\Delta^*(W) \supseteq T] = \text{info}_\Delta^*(T)$. $\square$

5.2 Galois Connections

A Galois connection between ordered classes $(P, \leq)$ and $(Q, \leq)$ is a pair (F, G) of antitone functions $F : P \rightarrow Q$ and $G : Q \rightarrow P$ such that $G \circ F$ and $F \circ G$ are extensive, i.e. $p \leq G(F(p))$ and $q \leq F(G(q))$ for all $p \in P$ and $q \in Q$. Propositions 11, 12 and 13 state basic results on Galois connections for ordered classes. Corresponding results for ordered sets are stated in Props. 6, 5 and 7, respectively, in the Formal Concept Analysis (FCA) textbook [16], and proofs apply without change (note however that Prop. 13 below states only one direction of its counterpart in the FCA textbook).

Proposition 11. *Let $(P, \leq)$ and $(Q, \leq)$ be ordered classes, and let $G : (Q, \leq) \rightarrow (P, \leq)$ be antitone. A function $F : P \rightarrow Q$ is a dual adjoint of G if and only if (F, G) is a Galois connection.*

Proposition 12. *Let (F, G) be a Galois connection of ordered classes. Then $F \circ G \circ F = F$ and $G \circ F \circ G = G$.*

Proposition 13. *Let $(P, \leq)$ and $(Q, \leq)$ be ordered classes with infima and suprema. Let $G : Q \rightarrow P$ be antitone. If G has a dual adjoint, then*

$$G\left(\bigvee_{i \in I} p_i\right) = \bigwedge_{i \in I} G(p_i). \quad (55)$$

Applying Prop. 13 to the Galois connection $(\text{info}_\Delta, \text{res}_\Delta)$ yields

$$\text{res}_\Delta\left(\bigvee_{i \in I} W_i\right) = \bigboxtimes_{i \in I} (\text{res}_\Delta(W_i)), \quad (56)$$

$$\text{info}_\Delta\left(\bigboxtimes_{i \in I} T_i\right) = \bigwedge_{i \in I} \text{info}_\Delta(T_i) \quad (57)$$

for all families $(W_i)_{i \in I}$ in WS_M and families $(T_i)_{i \in I}$ in Tab_G . Applying Prop. 13 to the Galois connection $(\text{info}_\Delta^*, \text{res}_\Delta^*)$ yields

$$\text{res}_\Delta^*(\bigvee_{i \in I} W_i) = \bigcap_{i \in I} \text{res}_\Delta^*(W_i) \quad (58)$$

$$\text{info}_\Delta^*(\bigcup_{i \in I} A_i) = \bigwedge_{i \in I} \text{info}_\Delta^*(A_i) \quad (59)$$

for all families $(W_i)_{i \in I}$ in WS_M and families $(A_i)_{i \in I}$ in $\mathfrak{P}(\text{Tup}(G))$.

We may write $T^\diamond$ instead of $\text{info}_\Delta(T)$ and $W^\diamond$ instead of $\text{res}_\Delta(W)$. Likewise, we may write $A^\bullet$ instead of $\text{info}_\Delta^*(A)$ and $W^\bullet$ instead of $\text{res}_\Delta^*(W)$. The use of the same symbol to denote either function of the Galois connection is a convention in FCA.

5.3 Concepts

We now define the concepts of a relational structure Δ with carrier set G and signature M . A *concept* of Δ is a pair (T, W) with $T \in \text{Tab}_G$, $W \in \text{WS}_M^{\text{rep}}$, $\text{res}_\Delta(W) = T$ and $\text{info}_\Delta(T) = W$. We say that T is the *extent* and W is the *intent* of the concept (T, W) . The set of all concepts of Δ is denoted by $\mathfrak{L}_\Delta$. It is characterized by the following proposition.

Proposition 14. *Let Δ be a relational structure. Then*

$$\mathfrak{L}_\Delta = \{(T^{\diamond\diamond}, T^\diamond) \mid T \in \text{Tab}_G\} \quad (60)$$

$$= \{(W^\diamond, W^{\diamond\diamond}) \mid W \in \text{WS}_M^{\text{rep}}\}. \quad (61)$$

Proof. By Prop. 12, we have $T^{\diamond\diamond\diamond} = T^\diamond$, so by definition $(T^{\diamond\diamond}, T^\diamond)$ is a concept. Conversely, if (T, W) is a concept, we obtain $(T, W) = (T^{\diamond\diamond}, T^\diamond)$ from $T^\diamond = W$ and $W^\diamond = T$. Eqns. (60), and (61) follow in the same way. $\square$

We say that (T_1, W_1) is a *subconcept* of (T_2, W_2) if

$$(T_1, W_1) \leq (T_2, W_2) :\Leftrightarrow T_1 \leq T_2, \quad (62)$$

which is equivalent to $W_2 \leq W_1$ because both functions of a Galois connection are antitone (cf. Section 5.2).

Theorem 4. *The ordered set $(\mathfrak{L}_\Delta, \leq)$ is a complete lattice. Infimum and supremum of a family $(T_i, W_i)_{i \in I}$ of concepts are given by*

$$\bigwedge_{i \in I} (T_i, W_i) = (\bigwedge_{i \in I} T_i, (\bigvee_{i \in I} W_i)^{\diamond\diamond}), \quad (63)$$

$$\bigvee_{i \in I} (T_i, W_i) = ((\bigvee_{i \in I} T_i)^{\diamond\diamond}, \bigwedge_{i \in I} W_i). \quad (64)$$

Proof. If $(\bigboxtimes_{i \in I} T_i, (\bigvee_{i \in I} W_i)^{\circ\circ})$ in (63) is indeed a concept, then by definition of the order in eqn. (62) it must be the infimum of $(T_i, W_i)_{i \in I}$. Using eqn. (56), we obtain $(\bigboxtimes_{i \in I} T_i, (\bigvee_{i \in I} W_i)^{\circ\circ}) = ((\bigvee_{i \in I} W_i)^\circ, (\bigvee_{i \in I} W_i)^{\circ\circ})$, which is a concept by (61). This proves eqn. (63), and eqn. (64) follows in the same way. Thus $(\mathfrak{L}_\Delta, \leq)$ is a complete lattice. $\square$

It is not possible to draw $(\mathfrak{L}_\Delta, \leq)$ as we did for the concept lattice in Fig. 2, because, as we will see, there is an infinite number of concepts. For finite Δ , we can draw lattices $\mathfrak{L}_\Delta[X]$ for a fixed set X of query variables, as introduced in the following.

A *concept of schema* X , for a given $X \subseteq \mathbf{var}$, is a pair (T, W) with $T \in G^X$ and $W \in \text{WS}_M^{\text{rep}}[X]$ that is stable under the pair of functions,

$$\text{res}_{\Delta, X} : \begin{cases} \text{WS}_M^{\text{rep}}[X] \rightarrow \mathfrak{P}(G^X) \\ (\mathfrak{A}, \lambda) \rightarrow \{\mu \in G^X \mid (\mathfrak{A}, \lambda) \lesssim (\Delta, \mu)\} \end{cases}, \quad (65)$$

$$\text{info}_{\Delta, X} : \begin{cases} \mathfrak{P}(G^X) \rightarrow \text{WS}_M^{\text{rep}}[X] \\ T \rightarrow \bigwedge_{\mu \in T} \text{rep}(\Delta, \mu) \end{cases}, \quad (66)$$

namely functions that satisfy $\text{res}_{\Delta, X}(T) = W$ and $\text{info}_{\Delta, X}(W) = T$. First, observe that $\text{res}_{\Delta, X}$ is the restriction of res_Δ to $\text{WS}_M^{\text{rep}}[X]$. Similarly, we have $\text{info}_{\Delta, X}(T) = \text{info}_\Delta(T)$ for all tables of schema X (i.e. for non-empty $T \in \mathfrak{P}(G^X)$). The schema of the empty table was defined to be $\mathbf{var}$, but $\text{info}_{\Delta, X}$ considers $\emptyset$ as a table of schema X , and consequently sends it to the greatest element in $\text{WS}_M^{\text{rep}}[X]$ instead (the empty infimum in (66)). In summary, the concepts of schema X are the concepts in $\mathfrak{L}_\Delta$ with extents in $\mathfrak{P}(G^X)$, except that the concept with an empty extent is assigned a different intent. Equivalently, the concepts of schema X are the concepts in $\mathfrak{L}_\Delta$ with intents in $\text{WS}_M^{\text{rep}}[X]$, with possibly an artificial bottom concept.

The set of concepts of schema X is denoted by $\mathfrak{L}_\Delta[X]$, and the order embedding,

$$\iota_X : \begin{cases} \mathfrak{L}_\Delta[X] \rightarrow \mathfrak{L}_\Delta \\ (T, W) \rightarrow \begin{cases} (T, W) & \text{if } T \neq \emptyset \\ (\emptyset, \text{info}_\Delta(\emptyset)) & \text{if } T = \emptyset \end{cases} \end{cases} \quad (67)$$

identifies $\mathfrak{L}_\Delta[X]$ with a subset of $\mathfrak{L}_\Delta$ (via concept extents). Concepts with extents in $\mathfrak{P}(G^X)$ are closed under non-empty infima (where the join in eqn. (63) simplifies to set intersection), and likewise, concepts with intents in $\text{WS}_M^{\text{rep}}[X]$ are closed under non-empty suprema (via the infimum on $\text{WS}_M^{\text{rep}}[X]$, cf. eqn. (64)). Considering eqn. (67), we can see that the respective infima and suprema exist in $\mathfrak{L}_\Delta[X]$, which makes $\mathfrak{L}_\Delta[X]$ a complete lattice (the top concept has extent G^X , and the bottom concept has intent $\text{info}_{\Delta, X}(\emptyset)$).

We call $\mathfrak{L}_\Delta[\{x_1, \dots, x_n\}]$ the lattice of n -ary concepts. The particular choice of variables is not significant, because for sets $X, Y \subseteq \mathbf{var}$ with $\#(X) = \#(Y)$, the lattices $\mathfrak{L}_\Delta[X]$ and $\mathfrak{L}_\Delta[Y]$ are isomorphic (as the meaning of queries is preserved under a one-to-one renaming of output variables). The unary concepts are arguably the most interesting ones, an example is presented in the next section.

5.4 Example: Lattice $\mathfrak{L}_\Delta[\{x_1\}]$ of Unary Concepts

Consider Δ in Fig. 10. The carrier set of Δ is $G = \{1, 2, 3, 4, 5, 6\}$, and the signature is $M = M_1 \cup M_2$ with $M_1 = \{a, b\}$ and $M_2 = \{r, s\}$. It is not possible to draw $\mathfrak{L}_\Delta$ in its entirety, because concepts exist for each arity, and there is an infinite number of arities.

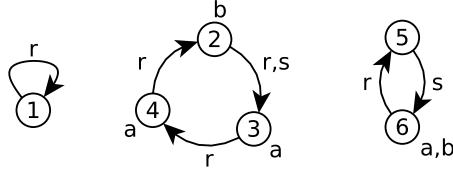


Fig. 10. Relational structure Δ .

However, since Δ is finite, the lattices $\mathfrak{L}_\Delta[\{x_1, \dots, x_n\}]$ of n -ary concepts are finite for all $n \in \mathbb{N}$ (because the possible extents are tables in Tab_G with schema $\{x_1, \dots, x_n\}$, and the number of such tables is finite). In the following, we examine the lattice $\mathfrak{L}_\Delta[\{x_1\}]$ of unary concepts, shown in Fig. 11. Its computation will be described in Section 5.5.

While the concepts in Fig. 2 were represented by nodes, we represent them in Fig. 11 by their concept intents, drawn as graphs. Figure 11 shows that Δ has 22 unary concepts. As in Fig. 2, a line between two concepts means that the lower concept is a subconcept of the upper concept, and no other concept is between them. It also means that there is a homomorphism from the upper graph to the lower graph (cf. the remark under eqn. (62)). The single yellow node, shown for each intent $(\mathfrak{A}, \lambda)$, is the query subject $\lambda(x_1)$, and must be preserved under homomorphism.

Six of the concepts in Fig. 11 are labeled with the object names $1, \dots, 6$. These are called *object concepts*, and their intents (the graphs themselves) are called *object intents*. The object concepts are the smallest concepts that have the respective object in their extent. The extent of a concept in Fig. 11 can be read as follows: a concept has g in its extent (for $g \in \{1, \dots, 6\}$) if it is connected to the object concept of g through downward lines (i.e. if the object concept is a subconcept). This labeling technique is known as reduced labeling (cf. [16]). For example, the extent of the top-most cycle in Fig. 11 is $\{1, 2, 3, 4\}$. The reader may verify that there is indeed a homomorphism from the cycle to the intents of the object concepts of 1,2,3 and 4, but not to those of 5 and 6.

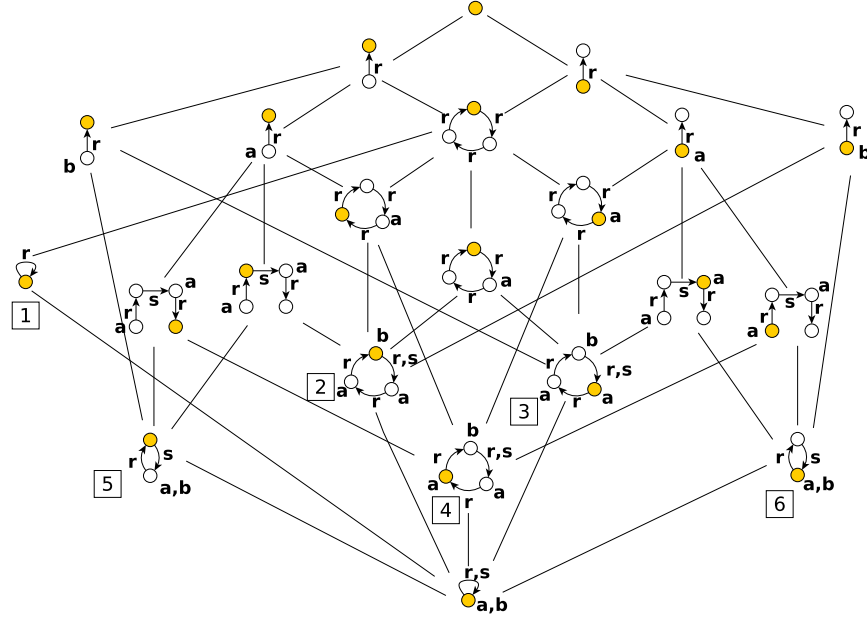


Fig. 11. Concept lattice with concept intents as graphs.

5.5 Naive computation of concepts

Recall by eqn. (60), the concepts of $\mathfrak{L}_\Delta$ are the pairs $(T^{\circ\circ}, T^\circ)$ for $T \in \text{Tab}_G$. For non-empty $T \subseteq G^X$, these are the concepts of $\mathfrak{L}_\Delta[X]$, with the possible exception of the bottom concept (for non-empty tables, eqns. (51) and (66) coincide). So if G and X are finite (and sufficiently small), we can compute $\mathfrak{L}_\Delta[X]$ by generating the intents T° for all non-empty $T \subseteq G^X$ (the intent of the bottom concept is always $(1_M, 1_X)$), and then solving these queries to obtain the extents.

The following proposition provides insight into what the intents are, and how they can be obtained.

Proposition 15. *Let $X \subseteq \mathbf{var}$ and $n \in \mathbb{N} \setminus \{0\}$. The concept intents generated from non-empty tables $T \subseteq G^X$ with at most n rows are, up to equivalence, the queries (Δ^n, λ) with $\lambda : X \rightarrow G^n$.*

Proof. Note that formally, a table $T \subseteq G^X$ does not specify a row order, and can not contain duplicate rows. We call a sequence $(\mu_0, \dots, \mu_{n-1}) \in (G^X)^n$, with repetitions allowed, an *ordered instance* of T , if $T = \{\mu_0, \dots, \mu_{n-1}\}$. The *instance intent* is defined as $(\mu_0, \dots, \mu_{n-1})^\star := \prod_{i=0}^{n-1} (\Delta, \mu_i)$, and we have

$$T^\circ \simeq (\mu_0, \dots, \mu_{n-1})^\star \quad (68)$$

for all instances $(\mu_0, \dots, \mu_{n-1})$ of a given T , because of $T^\circ = \bigwedge_{\mu \in T} \text{rep}(\Delta, \mu) = \bigwedge_{i=0}^{n-1} \text{rep}(\Delta, \mu_i) \simeq \prod_{i=0}^{n-1} (\Delta, \mu_i)$, cf. eqn. (47). Because every non-empty $T \subseteq G^X$

with at most n rows has an ordered instance in $(G^X)^n$ (duplicating some rows if necessary), the concept intents generated by such tables are precisely the instance intents of ordered instances in $(G^X)^n$.

To complete the proof, we show that the instance intents of ordered instances in $(G^X)^n$ are precisely the queries (Δ^n, λ) with $\lambda : X \rightarrow G^n$. The columns of $(\mu_0, \dots, \mu_{n-1}) \in (G^X)^n$ are specified by the function,

$$\lambda_{(\mu_0, \dots, \mu_{n-1})} : \begin{cases} X \rightarrow G^n \\ x \mapsto (\mu_0(x), \dots, \mu_{n-1}(x)) \end{cases}, \quad (69)$$

and we obtain $(\mu_0, \dots, \mu_{n-1})^* = \prod_{i=0}^{n-1} (\Delta, \mu_i) = (\Delta^n, \lambda_{(\mu_0, \dots, \mu_{n-1})})$ using Def. 1. Conversely, for a given $\lambda : X \rightarrow G^n$ we understand $\lambda(x)$ as the x -column of an ordered instance $(\mu_0, \dots, \mu_{n-1})$, where consequently the i -th row is defined $\mu_i(x) := (\lambda(x))_i$ for $x \in X$, and obtain $(\mu_0, \dots, \mu_{n-1})^* = (\Delta^n, \lambda)$. $\square$

We now illustrate how Prop. 15 is used to generate the concepts of $\mathfrak{L}_\Delta[\{x_1\}]$ in Fig. 11. According to Prop. 15, the intents generated by tables with 1 row are the queries (Δ, λ) with $\lambda : \{x_1\} \rightarrow G$. This produces six queries $(\Delta, \lambda_1), \dots, (\Delta, \lambda_6)$, where the query subjects $\lambda_1(x_1), \dots, \lambda_6(x_1)$ are the six nodes 1, $\dots$, 6 of Δ in Fig. 10. These are the six object intents in Fig. 11, drawn next to the respective object labels (the boxes in Fig. 11 with numbers in them).

The object intents contain all information available in Δ on the respective objects. The reader will notice that the intents (Δ, λ_i) are not drawn in their entirety: only the connected component which contains $\lambda_i(x_1)$ is shown. This would be expected, because the information expressed by the other components is not related to the subject $\lambda_i(x_1)$ (a formal treatment will be provided in Section 5.6).

The concept intents generated by tables with 2 rows are the queries (Δ^2, λ) with $\lambda : \{x_1\} \rightarrow G^2$. The second power Δ^2 is shown in Fig. 12. It consists of 25 nodes, which are all the possible query subjects, but of the 25 queries obtained from Δ^2 , some are equivalent. For example, the two circular queries with subjects $(3, 4)$ and $(4, 3)$ are isomorphic, and thus also equivalent. The same holds for the three queries with subjects $(1, 2)$, $(1, 3)$ and $(1, 4)$ that are obtained from the same component of Δ^2 (the lower right circle in Fig. 12). The queries with subjects $(1, 1), \dots, (6, 6)$ have already been seen in Δ . Discarding equivalences, we obtain 13 new intents from Δ^2 , which correspond to queries with the subjects $(2, 3)$, $(3, 4)$, $(4, 2)$, $(1, 5)$, $(1, 6)$, $(5, 6)$, $(5, 3)$, $(6, 2)$, $(6, 4)$, $(5, 2)$, $(6, 3)$, $(5, 4)$ and $(2, 1)$.

The two missing intents can be found near the top of Fig. 12. They can be obtained from Δ^3 , e.g. by choosing subjects $(4, 2, 5)$ and $(3, 4, 6)$. No further intents are obtained from Δ^3 . The fourth power Δ^4 produces no new intents at all. One may suspect that higher powers do not produce any further intents either. This is in fact the case as the following proposition establishes the terminating condition.

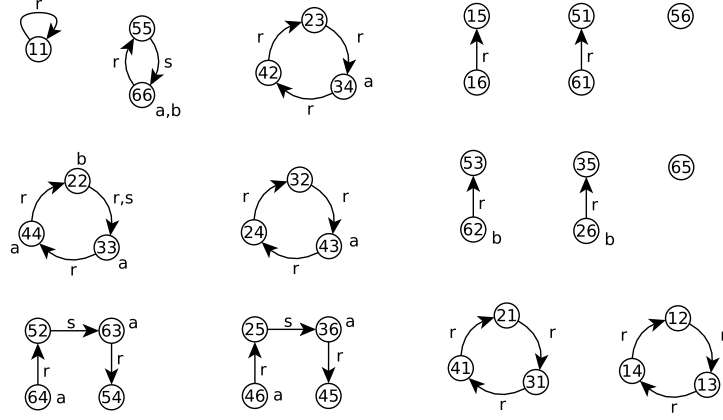


Fig. 12. Second power of Δ .

Proposition 16. Consider the following property for $k \in \mathbb{N}$:

$$P(k) := \text{“For all } T \subseteq G^X \text{ with } \#T = k \text{ there exists } S \subseteq G^X \text{ with } \#S < \#T \text{ and } S^\diamond = T^\diamond \text{”} \quad (70)$$

If $P(n)$ holds for a given $n \in \mathbb{N}$, then $P(n+1)$ holds as well.

Proof. Assume $P(n)$ holds. A table with $n+1$ rows can be written as $T \cup \{\mu\}$, where T has n rows. Let S denote a table with $\#S < \#T$ and $S^\diamond = T^\diamond$, then $\#(S \cup \{\mu\}) < \#(T \cup \{\mu\})$ and $(S \cup \{\mu\})^\diamond = S^\diamond \cap \{\mu\}^\diamond = T^\diamond \cap \{\mu\}^\diamond = (T \cup \{\mu\})^\diamond$.

5.6 The Support of a Query

In Fig. 11 we have identified each concept intent with the component that contains the query subject (cf. Section 5.5). To establish this formally, we define the *support* of a query as the union of connected components which contain one of the query subjects,

$$\text{supp}(\mathfrak{A}, \lambda) := (\mathfrak{A}|_{\bigcup_{a \in \text{rng}(\lambda)} C_{\mathfrak{A}}(a)}, \lambda), \quad (71)$$

where $C_{\mathfrak{A}}(a)$ denotes the component that contains a .

The following proposition states that concept intents can be compared by their support. This legitimates drawing only the support in Fig. 11.

Proposition 17. Let (T_1, W_1) and (T_2, W_2) be concepts of $\mathfrak{L}_\Delta$. Then

$$(T_1, W_1) \leq (T_2, W_2) \Leftrightarrow \text{supp}(W_2) \lesssim \text{supp}(W_1). \quad (72)$$

Proof. We set $(\mathfrak{A}_1, \lambda_1) := W_1$ and $(\mathfrak{A}_2, \lambda_2) := W_2$. If $(T_1, W_1) \leq (T_2, W_2)$, then there is a homomorphism $f : W_2 \rightarrow W_1$, and for all $x \in \text{def}(\lambda_2)$, it maps the component of $\lambda_2(x)$ in $\mathfrak{A}_2$ to the component of $\lambda_1(x)$ in $\mathfrak{A}_1$, so that $\text{supp}(W_2) \lesssim \text{supp}(W_1)$.

Now assume $\text{supp}(W_2) \lesssim \text{supp}(W_1)$. If $T_1 = \emptyset$, then by definition $(T_1, W_1) \leq (T_2, W_2)$. If $T_2 = \emptyset$, then (T_2, W_2) is the bottom concept, which satisfies $W_2 = \text{supp}(W_2)$, so that $W_2 \lesssim W_1$ by the assumption, and thus also $(T_1, W_1) \leq (T_2, W_2)$. Otherwise, we show that there is a homomorphism $f : \mathfrak{A}_2 \rightarrow \mathfrak{A}_1$. This homomorphism maps every component of W_2 that does *not* lie in $\text{supp}(W_2)$ to W_1 (for these components, no subject needs to be preserved). Considering eqn. (51) and Def. 1, we have $\mathfrak{A}_1 = \Delta^{T_1}$ and $\mathfrak{A}_2 = \Delta^{T_2}$. Since $T_2 \neq \emptyset$, there exists $t \in T_2$, and we have homomorphisms $\pi_t : \Delta^{T_2} \rightarrow \Delta$, defined by $\pi_t((x_\mu)_{\mu \in T_2}) := x_t$, and $\iota : \Delta \rightarrow \Delta^{T_1}$, defined by $\iota(x) := (x)_{t \in T_1}$, which provide $f := \iota \circ \pi_t$. $\square$

5.7 Set Representation of Extents

In the classical theory of Formal Concept Analysis [16, 15], extents are sets of objects, compared by set inclusion. Conversely, the extents of $\mathfrak{L}_\Delta$ are tables, and while for a fixed schema X , the table order coincides with set inclusion (as for the lattices $\mathfrak{L}_\Delta[X]$), this is no longer the case when comparing tables with differing schemas.

However, Prop. 4 establishes the view of tables as compact representations of partial assignment sets (which naturally arise from a logical perspective), such that the table order encodes set inclusion between these assignment sets. The functions *pset* and *table*, defined in eqns. (19) and (20), allow us to switch between table and set representations, and we apply them here at the level of concepts.

A *set-based concept* of Δ is a pair (A, W) with $A \subseteq \text{Tup}(G)$, $W \in \text{WS}_M^{\text{rep}}$, $\text{res}_\Delta^*(W) = A$ and $\text{info}_\Delta^*(A) = W$ (we have defined res_Δ^* and info_Δ^* in eqns. (17) and (50)). The set of all set-based concepts of Δ is denoted by $\mathfrak{L}_\Delta^*$.

Proposition 18. *The function*

$$f : \begin{cases} \mathfrak{L}_\Delta \rightarrow \mathfrak{L}_\Delta^* \\ (T, W) \mapsto (\text{pset}(T), W) \end{cases} \quad (73)$$

is an isomorphism of complete lattices.

Proof. The result set $\text{res}_\Delta^*(\mathfrak{A}, \lambda)$ of a query $(\mathfrak{A}, \lambda) \in \text{WS}_M$ contains all possible extensions of tuples in the result table $\text{res}_\Delta(\mathfrak{A}, \lambda)$, so formally we have

$$\mu \in \text{res}_\Delta^*(\mathfrak{A}, \lambda) \Leftrightarrow \mu|_{\text{def}(\lambda)} \in \text{res}_\Delta(\mathfrak{A}, \lambda) \quad (74)$$

for $\mu \in \text{Tup}(G)$. The functions *pset* and *table* allow to switch back and forth between both representations:

$$\text{res}_\Delta^*(\mathfrak{A}, \lambda) = \text{pset}(\text{res}_\Delta(\mathfrak{A}, \lambda)), \quad (75)$$

$$\text{res}_\Delta(\mathfrak{A}, \lambda) = \text{table}(\text{res}_\Delta^*(\mathfrak{A}, \lambda)). \quad (76)$$

By eqn.(61), the tables $\text{res}_\Delta(\mathfrak{A}, \lambda)$ are the extents of $\mathfrak{L}_\Delta$, and we obtain in the same way (using Prop. 12) that the sets $\text{res}_\Delta^*(\mathfrak{A}, \lambda)$ are the extents of $\mathfrak{L}_\Delta^*$. Then eqn. (75) shows that pset maps the extents of $\mathfrak{L}_\Delta$ to the extents of $\mathfrak{L}_\Delta^*$, surjectively. By Prop. 4, pset is also injective and order-preserving, so the function $f : \mathfrak{L}_\Delta \rightarrow \mathfrak{L}_\Delta^*$, defined by $f(T, W) := (\text{pset}(T), \text{info}_\Delta^*(\text{pset}(T)))$, is an isomorphism of complete lattices. It remains to be shown that this is the same f as in (73). Indeed, in the previous definition, we have

$$\begin{aligned} \text{info}_\Delta^*(\text{pset}(T)) &= \bigwedge_{\mu \in \text{pset}(T)} (\Delta, \mu) = \bigwedge_{\mu \in \text{pset}(T)} (\Delta, \mu|_{\text{schema}(T)}) \\ &= \bigwedge_{\mu \in T} (\Delta, \mu) = \text{info}_\Delta(T) = W, \end{aligned} \tag{77}$$

where the second equality follows from $(\Delta, \mu|_{\text{schema}(T)}) \lesssim (\Delta, \mu)$ and $\mu|_{\text{schema}(T)} \in \text{pset}(T)$. $\square$

6 Pattern Structures

Pattern structures, introduced by Ganter and Kuznetsov [15], provide a foundation for concepts with intents other than sets of attributes. A short summary is given in Section 2.2. In Section 6.1, we present pattern structures for the lattices $\mathfrak{L}_\Delta[X]$, and also for $\mathfrak{L}_\Delta^*$, thereby establishing the results of Section 5 as a special case of pattern structures. This allows the transfer of ideas, formulated for pattern structures, to conjunctive query concepts, and Sections 6.2 and 6.3 serve as examples for such a transfer of ideas.

6.1 Pattern Structures for Conjunctive Query Concepts

In Section 4.3, we have turned the preordered class $(\text{WS}_M, \lesssim)$ of patterns into an ordered class $(\text{WS}_M^{\text{rep}}, \leq)$, thereby satisfying one condition for pattern structures, but strictly speaking, the patterns must be contained in an ordered set. The idea for turning $(\text{WS}_M^{\text{rep}}, \leq)$ into an ordered set is to keep only those patterns up to a certain size. More precisely, we define

$$\text{WS}_{M,\alpha} := \{W \in \text{WS}_M \mid \#|\mathfrak{A}| < \alpha\}, \tag{78}$$

$$\text{WS}_{M,\alpha}^{\text{rep}} := \{\text{rep}(W) \mid W \in \text{WS}_{M,\alpha}\} \tag{79}$$

for an infinite cardinal α . An important special case is $\alpha = \aleph_0$, then $\#|\mathfrak{A}| < \alpha$ simply means that the carrier set $|\mathfrak{A}|$ is finite.

Proposition 19. *For every infinite cardinal α , the pair $(\text{WS}_{M,\alpha}^{\text{rep}}, \wedge)$ is a meet-semilattice.*

Proof. We have to show that $\text{WS}_{M,\alpha}^{\text{rep}}$ is closed under $\wedge$. Let $W_1, W_2 \in \text{WS}_{M,\alpha}^{\text{rep}}$, then $W_1 \simeq (\mathfrak{A}_1, \lambda_1)$ and $W_2 \simeq (\mathfrak{A}_2, \lambda_2)$ for some $(\mathfrak{A}_1, \lambda_1), (\mathfrak{A}_2, \lambda_2) \in \text{WS}_M$ with $\#|\mathfrak{A}_1|, \#|\mathfrak{A}_2| < \alpha$. Thus $W_1 \wedge W_2 \simeq (\mathfrak{A}_1, \lambda_1) \times (\mathfrak{A}_2, \lambda_2)$, and by (30), the carrier set

of the product is $|\mathfrak{A}_1| \times |\mathfrak{A}_2|$. If both $\#|\mathfrak{A}_1|$ and $\#|\mathfrak{A}_2|$ are finite, then $\#(|\mathfrak{A}_1| \times |\mathfrak{A}_2|)$ is finite, and thus $\#(|\mathfrak{A}_1| \times |\mathfrak{A}_2|) < \alpha$. Otherwise, WLOG $\#|\mathfrak{A}_1| \leq \#|\mathfrak{A}_2|$ and thus $\#(|\mathfrak{A}_1| \times |\mathfrak{A}_2|) \leq \#(|\mathfrak{A}_2| \times |\mathfrak{A}_2|) = \#|\mathfrak{A}_2| < \alpha$, where the middle equation follows by Hessenberg's theorem [9]. Therefore, $W_1 \wedge W_2 \in \text{WS}_{M,\alpha}^{\text{rep}}$. $\square$

Accordingly, we define $\text{WS}_{M,\alpha}[X] := \text{WS}_{M,\alpha} \cap \text{WS}_M[X]$, and $\text{WS}_{M,\alpha}^{\text{rep}}[X] := \{\text{rep}(W) \mid W \in \text{WS}_{M,\alpha}[X]\}$ defines the meet-subsemilattice of queries in X .

Proposition 20. *Let Δ be a relational structure. For sufficiently large α , the triple $(G^X, (\text{WS}_{M,\alpha}^{\text{rep}}[X], \wedge), \delta)$ with $\delta(\mu) := \text{rep}(\Delta, \mu)$ is a pattern structure, and its concept lattice is $\mathfrak{L}_\Delta[X]$.*

Proof. Choose α such that $\bigwedge_{\mu \in A} \delta(\mu) \in \text{WS}_{M,\alpha}^{\text{rep}}[X]$ for all $A \subseteq G^X$, then by definition, $(G^X, (\text{WS}_{M,\alpha}^{\text{rep}}[X], \wedge), \delta)$ is a pattern structure (cf. Section 2.2). Applying the generic definition of the Galois connection, given in eqns. (3) and (4), produces eqns. (65) and (66), using $(\mathfrak{A}, \lambda) \leq \text{rep}(\Delta, \mu) \Leftrightarrow (\mathfrak{A}, \lambda) \lesssim (\Delta, \mu)$ for (65). The pattern concepts are the stable pairs under eqns. (65) and (66), and by definition, these pairs are the elements of $\mathfrak{L}_\Delta[X]$. $\square$

Extents of pattern concepts are sets (ordered by inclusion), so strictly speaking, $\mathfrak{L}_\Delta$ can not be obtained within the framework of pattern structures. However, Prop. 18 shows that $\mathfrak{L}_\Delta$ can be identified with $\mathfrak{L}_\Delta^*$, and we can state a pattern structure for $\mathfrak{L}_\Delta^*$.

Proposition 21. *Let Δ be a relational structure. For sufficiently large α , the triple $(\text{Tup}(G), (\text{WS}_{M,\alpha}^{\text{rep}}, \wedge), \delta)$ with $\delta(\mu) := \text{rep}(\Delta, \mu)$ is a pattern structure, and its concept lattice is $\mathfrak{L}_\Delta^*$.*

Proof. Choose α such that $\bigwedge_{\mu \in A} \delta(\mu) \in \text{WS}_{M,\alpha}^{\text{rep}}$ for all $A \subseteq \text{Tup}(G)$, then by definition, $(\text{Tup}(G), (\text{WS}_{M,\alpha}^{\text{rep}}, \wedge), \delta)$ is a pattern structure (cf. Section 2.2). Applying the generic definition of the Galois connection, given in eqns. (3) and (4), produces eqns. (17) and (50), using $(\mathfrak{A}, \lambda) \leq \text{rep}(\Delta, \mu) \Leftrightarrow (\mathfrak{A}, \lambda) \lesssim (\Delta, \mu)$ for eqn. (17). The pattern concepts are the stable pairs under eqns. (17) and (50), and by definition, these pairs are the elements of $\mathfrak{L}_\Delta^*$. $\square$

6.2 Kernel Operators

The patterns which represent concept intents can be large and complex, even in their minimized form, so that their computation becomes infeasible, or they can no longer be easily understood or represented. Since concept intents simply reflect what information is available on a set of objects or tuples, the key to solving the complexity problem lies in the reduction of information.

Ganter and Kuznetsov [15] address complexity for pattern structures by kernel operators (also called *projections*) which map patterns to simpler patterns. A kernel operator on a partially ordered set $(P, \leq)$ is a map $\kappa : P \rightarrow P$ associated with a *kernel range* $K \subseteq P$ such that,

$$\kappa(x) = \max\{y \in K \mid y \leq x\} \quad (80)$$

for all $x \in P$, where $\kappa(x)$ is called the *kernel* of x . The kernel is the closest approximation of x by a weaker pattern in K , and consequently $\kappa(x) = x$ for all $x \in K$. Kernel operators are the order-theoretic dual of closure operators, for their theoretic properties see e.g. [11].

Lemma 1. *Let κ be a kernel operator on $(P, \leq)$ with kernel range K . For all $A \subseteq P$, if A has an infimum q in P , then $\kappa(A)$ has the infimum $\kappa(q)$ in K .*

Proof. Let q be the infimum of $A \subseteq P$. The elements of $\kappa(A)$ are the kernels $\kappa(p)$ with $p \in A$, and from $q \leq p$ we obtain $\kappa(q) \leq \kappa(p)$ (cf. (80)), so $\kappa(q)$ is a lower bound of $\kappa(A)$. If $k \in K$ is another lower bound of $\kappa(A)$, then $k \leq \kappa(p) \leq p$ for all $p \in A$, which implies $k \leq q$ and thus $k = \kappa(k) \leq \kappa(q)$. So $\kappa(q)$ is the greatest lower bound of $\kappa(A)$ in K . $\square$

Proposition 22. *Let $(P, \wedge)$ be a meet-semilattice, and κ a kernel operator on P with kernel range K . Then $(K, \wedge_K)$ with*

$$k_1 \wedge_K k_2 := \kappa(k_1 \wedge k_2) \quad (81)$$

for all $k_1, k_2 \in K$ is a meet-semilattice.

Proof. This follows from Lemma 1 with $A = \{k_1, k_2\}$. $\square$

Proposition 23. *Let $(E, (P, \wedge), \delta)$ be a pattern structure, and κ a kernel operator on P with kernel range K . Then $(E, (K, \wedge_K), \kappa \circ \delta)$ is a pattern structure.*

Proof. We have to show that, for all $A \subseteq E$, the infimum of $\kappa(\delta(A))$ exists in K . So let $A \subseteq E$. Because $(E, (P, \wedge), \delta)$ is a pattern structure, the infimum $q := \bigwedge \delta(A)$ exists in P , and by Lemma 1, $\kappa(q)$ is the infimum of $\kappa(\delta(A))$ in K . $\square$

Kernel operators for conjunctive queries are most naturally defined in the pre-order (i.e. on the queries themselves), and not on the representatives (i.e. equivalence classes), as required by Prop. 23. As a practical approach, we define a kernel operator by a function $\kappa : \text{WS}_{M,\alpha}[X] \rightarrow \text{WS}_{M,\alpha}[X]$ that assigns to each $W \in \text{WS}_{M,\alpha}[X]$ a greatest lower bound $\kappa(W)$ in a given subset $K \subseteq \text{WS}_{M,\alpha}[X]$. We do not require κ to be idempotent (in the sense that $\kappa(\kappa(W)) = \kappa(W)$) or surjective on K .

Proposition 25 states that every such κ induces a kernel operator κ^{rep} on $\text{WS}_{M,\alpha}^{rep}[X]$. Before covering details, we present three kernel operators for finite queries with a single subject, defined by functions $\kappa : \text{WS}_{M,\aleph_0}[\{x_1\}] \rightarrow \text{WS}_{M,\aleph_0}[\{x_1\}]$. In all cases, M is assumed to contain only unary and binary relation symbols.

k -cut The k -cut of a graph is the subgraph that consists of the nodes that can be reached from $\lambda(x_1)$ by traversing at most k edges (in either direction). The subgraph is induced, i.e. all edges between the nodes are included.

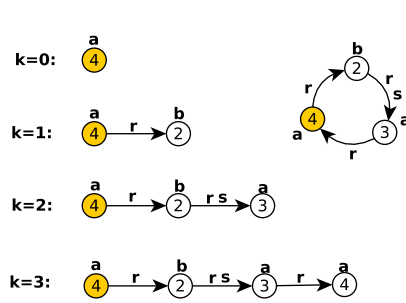


Fig. 13. Directed unfolding of the top right graph for $k = 0, 1, 2, 3$.

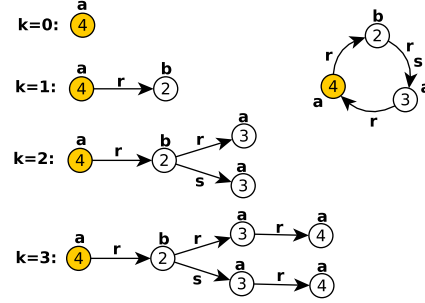


Fig. 14. Directed unraveling of the top right graph for $k = 0, 1, 2, 3$.

directed k -unfolding The *directed k -unfolding* of a graph (cf. Fig. 13) is a tree with root $\lambda(x_1)$ where all outgoing edges and their target nodes are replaced by disjoint copies (thus eliminating cycles), successively up to a maximum tree depth of k . All incoming edges, and outgoing edges at depth greater than k , are discarded.

directed k -unraveling The *directed k -unraveling* of a graph (cf. Fig. 14) is defined like the directed k -unfolding, except that an edge with multiple labels $m_1, \dots, m_n$ is split into n edges, each carrying one of the labels, and with its own copy of the target.

In each example, a query is projected into a subset of patterns, defined respectively by the properties

$$P_1(W) := \text{"}W \text{ has radius at most } k\text{"}, \quad (82)$$

$$P_2(W) := \text{"}W \text{ is a rooted tree of depth at most } k\text{"}, \quad (83)$$

$$P_3(W) := \text{"}W \text{ is a thin rooted tree of depth at most } k\text{"}, \quad (84)$$

where the *radius* of $W \in \text{WS}_M[\{x_1\}]$ is defined as the smallest $k \in \mathbb{N}$ such that every node can be reached from $\lambda(x_1)$ by traversing at most k edges (in either direction), and a rooted tree is called *thin* if each edge is labeled by exactly one symbol.

Proposition 24. *The k -cut, directed k -unfolding and directed k -unraveling provide greatest lower bounds in the subsets defined by properties P_1 , P_2 and P_3 , respectively.*

Proof. Let $\kappa_1(W)$ denote the k -cut of W . The subgraph embedding $\iota : \kappa_1(W) \rightarrow W$ provides $\kappa_1(W) \lesssim W$. For $U \lesssim W$ with radius at most k , there exists $f : U \rightarrow W$, and the image $f(U)$ is contained in $\kappa_1(W)$, thus $U \lesssim \kappa_1(W)$. So $\kappa_1(W)$ is a greatest lower bound of W in the set of graphs with radius at most k .

Let $\kappa_2(W)$ denote the directed k -unfolding of W . Each node of $\kappa_2(W)$ is identified with the unique path from the root to that node, and a homomorphism

$f : \kappa_2(W) \rightarrow W$ is then given by $f(x_0, \dots, x_i) := x_i$, cf. Fig. 13. Let T be a rooted tree with $T \lesssim W$ and depth at most k . Because of $T \lesssim W$, there is a homomorphism $f : T \rightarrow W$, and as with $\kappa_2(W)$, we identify each node of T with the unique path from the root to that node. Then $\tilde{f}(y_0, \dots, y_j) := (f(y_0), \dots, f(y_j))$ defines a homomorphism $\tilde{f} : T \rightarrow \kappa_2(W)$, thus $T \lesssim \kappa_2(W)$, which proves the greatest lower bound. The same proof applies to k -unraveling, except that paths are modeled by alternating sequences $(x_0, r_1, x_1, \dots, r_i, x_i)$, which include the edge labels.

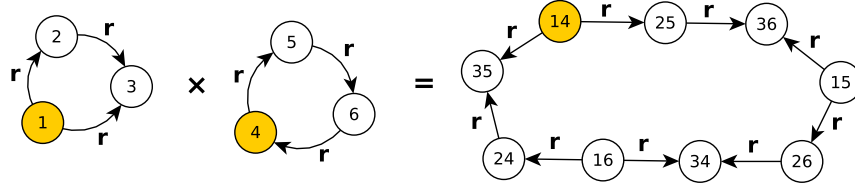


Fig. 15. The subset of queries with radius ≤ 1 is not closed under infima.

Proposition 25. *Let $\kappa : \text{WS}_{M,\alpha}[X] \rightarrow \text{WS}_{M,\alpha}[X]$ be a function such that $\kappa(x)$ is a greatest lower bound of x in a given subset $K \subseteq \text{WS}_{M,\alpha}[X]$. Then $\kappa^{rep} := \text{rep} \circ \kappa$ is a kernel operator on $\text{WS}_{M,\alpha}^{rep}[X]$, its kernel range is $K^{rep} := \{\text{rep}(W) \mid W \in K\}$, and*

$$\kappa(W) \simeq \kappa^{rep}(\text{rep}(W)) \quad (85)$$

$$\kappa(W_1 \times W_2) \simeq \text{rep}(W_1) \wedge_{K^{rep}} \text{rep}(W_2) \quad (86)$$

for all $W, W_1, W_2 \in \text{WS}_{M,\alpha}[X]$.

Proof. Equivalent queries W_1 and W_2 have the same lower bounds in K , so that the greatest lower bounds $\kappa(W_1)$ and $\kappa(W_2)$ must be equivalent,

$$W_1 \simeq W_2 \Rightarrow \kappa(W_1) \simeq \kappa(W_2). \quad (87)$$

So from $W \simeq \text{rep}(W)$ we obtain $\kappa(W) \simeq \kappa(\text{rep}(W)) \simeq \kappa^{rep}(\text{rep}(W))$, which shows (85). For $x \in \text{WS}_{M,\alpha}^{rep}[X]$ there exists $\tilde{x} \in \text{WS}_{M,\alpha}[X]$ with $\text{rep}(\tilde{x}) = x$, and considering (85), we have $\kappa^{rep}(x) \simeq \kappa(\tilde{x}) \in K$. This shows that $\kappa^{rep}(x) \in K^{rep}$. Considering (80), it remains to be shown that

$$y \leq x \Leftrightarrow y \leq \kappa^{rep}(x) \quad (88)$$

for all $x \in \text{WS}_{M,\alpha}^{rep}[X]$ and $y \in K^{rep}$. For such x and y , we have $\tilde{x} \in \text{WS}_{M,\alpha}[X]$ and $\tilde{y} \in K$ with $\text{rep}(\tilde{x}) = x$ and $\text{rep}(\tilde{y}) = y$. Because $\kappa(\tilde{x})$ is a greatest lower bound of $\tilde{x}$ in K , we have $\tilde{y} \lesssim \tilde{x} \Leftrightarrow \tilde{y} \lesssim \kappa(\tilde{x})$, and applying (46) to both sides

yields (88). This shows that κ^{rep} is a kernel operator with kernel range K^{rep} . By (47), we have $W_1 \times W_2 \simeq rep(W_1) \wedge rep(W_2)$, whence (87) and (81) provide $\kappa(W_1 \times W_2) \simeq \kappa(rep(W_1) \wedge rep(W_2)) = rep(W_1) \wedge_{K^{rep}} rep(W_2)$, showing (86). $\square$

A pattern structure that generates $\mathfrak{L}_\Delta[X]$ was stated in Prop. 20. For any kernel operator κ , we obtain another pattern structure as in Prop. 23, and use $\mathfrak{L}_\Delta^\kappa[X]$ to denote its concept lattice. Proposition 26 shows that $\mathfrak{L}_\Delta^\kappa[X]$ can be identified with an $\wedge$ -sublattice of $\mathfrak{L}_\Delta[X]$. In this context, we use $(\cdot)^\diamond$ as a shorthand for (65) and (66) defining $\mathfrak{L}_\Delta[X]$, and $(\cdot)^\star$ for the corresponding functions of $\mathfrak{L}_\Delta^\kappa[X]$, as stated generically in (3) and (4).

Proposition 26. *Let $(G, (P, \wedge), \delta)$ be a pattern structure for $\mathfrak{L}_\Delta[X]$, and κ a kernel operator on P with kernel range K . The function*

$$f : \begin{cases} \mathfrak{L}_\Delta^\kappa[X] \rightarrow \mathfrak{L}_\Delta[X] \\ (A, k) \mapsto (A, A^\diamond) \end{cases} \quad (89)$$

is an infimum-preserving order embedding, and

$$g : \begin{cases} \mathfrak{L}_\Delta[X] \rightarrow \mathfrak{L}_\Delta^\kappa[X] \\ (A, d) \mapsto (\kappa(d)^\star, \kappa(d)) \end{cases} \quad (90)$$

is its left inverse, in particular g is surjective.

Proof. First, we relate the two Galois connections. For all kernels $k \in K$,

$$k^\star = \{\mu \in G^X \mid k \leq \kappa(\delta(\mu))\} = \{\mu \in G^X \mid k \leq \delta(\mu)\} = k^\diamond, \quad (91)$$

where in the middle equation, “ $\subseteq$ ” follows from $\kappa(\delta(\mu)) \leq \delta(\mu)$, and “ $\supseteq$ ” follows directly from (80). Similarly, for all $A \subseteq G^X$, we obtain

$$\kappa(A^\diamond) = \kappa(\bigwedge \{\delta(\mu) \mid \mu \in A\}) = \bigwedge_K \{\kappa(\delta(\mu)) \mid \mu \in A\} = A^\star, \quad (92)$$

where the middle equation follows from Lemma 1.

For $(A, k) \in \mathfrak{L}_\Delta^\kappa[X]$, we have $A = k^\star$ and obtain, using (91), that $f(A, k) = (k^\star, k^\diamond) = (k^\diamond, k^\diamond) \in \mathfrak{L}_\Delta[X]$, which means that f is well-defined. Similarly, for $(A, d) \in \mathfrak{L}_\Delta[X]$, we have $d = A^\diamond$ and obtain, using (92), that $g(A, d) = ((\kappa(A^\diamond))^\star, \kappa(A^\diamond)) = (A^\star, A^\star) \in \mathfrak{L}_\Delta^\kappa[X]$, so g is well-defined. Since f identifies concepts by their extents, it is an order embedding and preserves infima, which are given by set intersection (cf. Prop. 1). Finally, using (92) again we have $g(f(A, k)) = g(A, A^\diamond) = (\kappa(A^\diamond)^\star, \kappa(A^\diamond)) = (A^\star, A^\star) = (A, k)$, which means that g is a left inverse of f , and thus also surjective. $\square$

6.3 Next Concept Algorithm

This section describes how the concepts of $\mathfrak{L}_\Delta^\kappa[\{x_1\}]$ (for finite Δ) can be generated with a certain variant of Ganter’s “Next Concept” algorithm [14], mentioned in the introductory paper on pattern structures [15, p. 132f.]. The CbO

algorithm by Kuznetsov [26] provides an explicit formulation. A key feature of our variant is that it allows us to compute each new concept intent as a product of exactly two previously computed intents, rather than directly from an object or tuple set as per eqn. (4). The significance for conjunctive queries is that this prevents concept intents from becoming unnecessarily large, crucial when computing homomorphisms. The algorithm in Fig. 16 restates the approach in the context of our work. It is explained below, and its equivalence to the “Next Concept” algorithm proven in Props. 27 and 28.

```

1  function concepts( $C, k$ ):
2       $L := [C]$ 
3      for  $i = n, n - 1, \dots, k + 1$ :
4           $A := \text{extent}(C)$ 
5          if  $g_i \in A$ :
6              continue
7           $W := \text{intent}(C) \times \delta(g_i)$ 
8           $W := \text{kernel}(W)$ 
9           $W := \text{minimize}(W)$ 
10          $E := \text{result}(W)$ 
11         if  $E_{<i} == A_{<i}$ :
12              $L += \text{concepts}((E, W), i)$ 
13     return  $L$ 
14
15 concepts( $\perp, 0$ )

```

Fig. 16. Next Concept algorithm (variant).

The extents of $\mathfrak{L}_\Delta^\kappa[\{x_1\}]$ are tables with a single column, and for simplicity, we identify them with subsets of the carrier set G of Δ . The Next Concept algorithm assumes an (arbitrary) enumeration of objects, i.e. $G = \{g_1, \dots, g_n\}$ with $g_1, \dots, g_n$ distinct. Objects are then ordered by their indices, i.e. $g_1 < \dots < g_n$. A subset $A \subseteq G$ corresponds to an n -bit word, where the i -th bit is set, if and only if $g_i \in A$. For example, if $n = 12$, the subset $\{g_2, g_4, g_6, g_7, g_{10}, g_{11}, g_{12}\}$ corresponds to the 12-bit word 010101100111 in Fig. 17. This establishes a total order on $\mathfrak{P}(G)$, called the *lectic order* on $\mathfrak{P}(G)$, where subsets of G are compared by their bit representations (read as binary numbers), or formally, $A < B :\Leftrightarrow \min(B \setminus A) < \min(A \setminus B)$ for $A, B \in \mathfrak{P}(G)$.

The Next Concept algorithm computes all concept extents in lectic order, starting with the smallest extent. We shall write $\text{cl}(A)$ to denote the concept extent generated by $A \subseteq G$ (which is $A^\bullet$ when computing $\mathfrak{L}_\Delta^\kappa[\{x_1\}]$). The smallest extent in the lectic order is then given as $\text{cl}(\emptyset)$. Note that cl is a closure operator, i.e. it is monotone, idempotent and extensive (cf. Section 5.2). For

$\{g_2, g_4, g_6, g_7, g_{10}, g_{11}, g_{12}\}$	7												
$\{g_2, g_4, g_6, g_{10}, g_{12}\}$	4	1	2			1	3			2	3	2	
$\{g_2, g_6\}$	2	0	1	0	1	1	0	0	1	1	1		
$\emptyset$	0	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}
Stack		Binary word											

Fig. 17. Stack and binary word

$A \subseteq G$ and $i \in \{1, \dots, n\}$, we define

$$A_{<i} := A \cap \{g_1, \dots, g_{i-1}\}, \quad (93)$$

$$A \oplus i := \text{cl}(A_{<i} \cup \{g_i\}). \quad (94)$$

It is shown in [16] that the extent which follows an extent $A \subseteq G$ in the lexic order is $A \oplus i$ for the largest i such that $g_i \notin A$ and $(A \oplus i)_{<i} = A_{<i}$.

The `concepts()` function in Fig. 16 is given the bottom concept $\perp$ (line 15), computes all successors in lexic order, and returns them in the list L (line 13). At any time, the current concept (for which the successor is computed) is the last element of L . When a new concept has been created, the value of $A \oplus i$ is computed from $i = n$ downwards (line 3), and stored in the variable E (line 10), until the greatest i has been found which satisfies both $g_i \notin A$ (lines 5–6) and $(A \oplus i)_{<i} = A_{<i}$ (line 11). Once the successor is obtained, it is passed to another instance of `concepts()` (line 12). Note that the for-loop terminates at $i = k$ instead of going down to 1. This simply means that, for the purpose of computing $A \oplus i$ for $i \leq k$, the extent A can be replaced with a calling instance's A (see Prop. 28), so control is returned to the caller. Accordingly, when the called instance in line 12 returns, the for-loop continues to compute $A \oplus i$ for decreasing i , in search of the current concept's successor.

To prove that the algorithm in Fig. 16 performs the same steps as the Next Concept algorithm, we define the *canonical generator* of an extent A as

$$\text{gen}(A) := \{g_i \in A \mid g_i \notin \text{cl}(A_{<i})\}. \quad (95)$$

Proposition 27 shows that $\text{gen}(A)$ is indeed a generator of A .

Proposition 27. *Let A be an extent, then $\text{cl}(\text{gen}(A)) = A$.*

Proof. It is clear that $\text{gen}(\text{cl}(A)) \subseteq A$. Suppose that $\text{cl}(\text{gen}(A)) \neq A$, then $g_j = \min(A \setminus \text{cl}(\text{gen}(A)))$ exists, and $A_{<j} = (\text{cl}(\text{gen}(A)))_{<j}$ because g_j is chosen minimal. From $g_j \notin \text{gen}(A)$ follows $g_j \in \text{cl}(A_{<j})$ using (95), so that $g_j \in \text{cl}(A_{<j}) = \text{cl}((\text{cl}(\text{gen}(A)))_{<j}) \subseteq \text{cl}(\text{cl}(\text{gen}(A))) = \text{cl}(\text{gen}(A))$, contradiction! $\square$

Proposition 28 provides an inductive definition of the canonical generator over the next concept operation (as used by the Next Concept algorithm).

Proposition 28.

$$\text{gen}(\text{cl}(\emptyset)) = \emptyset \quad (96)$$

$$\text{gen}(A \oplus i) = (\text{gen}(A))_{<i} \cup \{g_i\} \quad (97)$$

for all closures A with $(A \oplus i)_{<i} = A_{<i}$ and $g_i \notin A$.

Proof. From eqn. (95), we obtain $\text{gen}(\text{cl}(\emptyset)) = \{g_i \in \text{cl}(\emptyset) \mid g_i \notin \text{cl}((\text{cl}(\emptyset))_{<i})\}$, where $\text{cl}(\emptyset) \subseteq \text{cl}((\text{cl}(\emptyset))_{<i}) \subseteq \text{cl}(\text{cl}(\emptyset)) = \text{cl}(\emptyset)$, which shows eqn. (96). For eqn. (97), we show $g_j \in \text{gen}(A \oplus i) \Leftrightarrow g_j \in (\text{gen}(A))_{<i} \cup \{g_i\}$ for all $j \in \{1, \dots, n\}$. For $j < i$, we have

$$\begin{aligned} g_j \in \text{gen}(A \oplus i) &\Leftrightarrow g_j \in A \oplus i \wedge g_j \notin \text{cl}((A \oplus i)_{<j}) \\ &\Leftrightarrow g_j \in A \wedge g_j \notin \text{cl}(A_{<j}) \\ &\Leftrightarrow g_j \in \text{gen}(A), \end{aligned} \quad (98)$$

where the second equivalence follows from the assumption $(A \oplus i)_{<i} = A_{<i}$. It remains to be shown that $g_i \in \text{gen}(A \oplus i)$ and $g_j \notin \text{gen}(A \oplus i)$ for $j > i$, cf. eqn. (97). Suppose that $g_i \notin \text{gen}(A \oplus i)$. Using (95) and the assumption $(A \oplus i)_{<i} = A_{<i}$, we obtain $g_i \in \text{cl}((A \oplus i)_{<i}) = \text{cl}(A_{<i}) \subseteq A$, which contradicts the other assumption $g_i \notin A$, so the supposition is wrong and $g_i \in \text{gen}(A \oplus i)$. Now let $j > i$. Then $A_{<i} \cup \{g_i\} \subseteq \{g_1, \dots, g_{j-1}\}$, and considering eqn. (94), we obtain $A_{<i} \cup \{g_i\} \subseteq (A \oplus i)_{<j}$. Thus $A \oplus i = \text{cl}(A_{<i} \cup \{g_i\}) \subseteq \text{cl}((A \oplus i)_{<j})$, which means by eqn. (95) that $g_j \notin \text{gen}(A \oplus i)$. $\square$

Figure 17 shows a call stack for the `concepts()` function. For each instance, it shows the values C (only the extent) and k it was called with. For each extent on the stack, the indices at the level and below that extent (excluding 0) describe a generator for that extent. With respect to Fig. 17, this means $\text{cl}(\emptyset) = \emptyset$, $\text{cl}(\{g_2\}) = \{g_2, g_6\}$, $\text{cl}(\{g_2, g_4\}) = \{g_2, g_4, g_6, g_{10}, g_{12}\}$ and $\text{cl}(\{g_2, g_4, g_7\}) = \{g_2, g_4, g_6, g_7, g_{10}, g_{11}, g_{12}\}$. When the algorithm starts, the concept extent is $\text{cl}(\emptyset)$ and the corresponding generator is $\emptyset$, as in eqn. (96). Whenever a new extent A is put on the stack, the algorithm modifies the stack according to the right-hand side of (97), and the left-hand side of (97) shows that the computed concept is the successor in the lectic order.

7 Concept Graphs

In the early 1990's [37], Rudolf Wille started a program of work he named *restructuring mathematical logic* [35]. Wille's research focused on the mathematization of traditional philosophical logic following C.S. Peirce [30] – whose work informed early thinking in modern mathematical logic – with G.D. Birkhoff's work on lattice and order theory [6].

The Peircean philosophical origin has as its basis the notions of *concept*, *judgment* and *conclusion*. A judgment, in this context, is a composition of concepts akin to a statement. A formalization of concepts was already achieved in Formal

Concept Analysis, and after some earlier considerations [35], Wille introduced *concept graphs* [36], which derive from J.F. Sowa's Conceptual Graphs [33] which in turn are based on Peirce's Existential Graphs [30], as a formalization of judgments. *Power context families* were introduced by Wille in [36] as a relational data model within which concept graphs are defined. In this section, we discuss these notions in the context of the results of the previous sections.

7.1 Power Context Families

Power context families are introduced by Wille [36] as a representation of relational data based on the formal contexts of Formal Concept Analysis.

Definition 3. A power context family is a sequence $(\mathbb{K}_k)_{k \geq 1}$ of formal contexts $\mathbb{K}_k = (G_k, M_k, I_k)$ such that $G_k \subseteq (G_1)^k$ for all $k \geq 2$.

The context $\mathbb{K}_1$ in the above definition is called the *object context* of the power context family, and it is thought of as a formal context as in Fig. 1, where objects are described by attributes. The other contexts are called *relation contexts*. The objects of $\mathbb{K}_k$, $k \geq 2$, are k -tuples of objects in G_1 , and each attribute expresses a k -ary relation on G_1 . Fig. 19 shows a power context family $(\mathbb{K}_1, \mathbb{K}_2)$ with a single relation context (for the binary relations).

The attributes of a power context family define a relational signature M , where the set M_k of k -ary relation symbols coincides with the attribute set of $\mathbb{K}_k$, $k \geq 1$. We then say that $\vec{\mathbb{K}}$ is a power context family over the signature M . A power context family $\vec{\mathbb{K}}$ over M can be expressed by an M -structure $\Delta_{\vec{\mathbb{K}}}$, given by

$$|\Delta_{\vec{\mathbb{K}}}| := G_1, \quad (99)$$

$$m^{\Delta_{\vec{\mathbb{K}}}} := m^{I_k} \quad (100)$$

for all $m \in M_k$ and $k \geq 1$, where we write m^{I_k} instead of m' or $\{m\}'$ (cf. (1)) to avoid using the same notation for all contexts in the power context family. Fig. 18 shows the relational structure $\Delta_{\vec{\mathbb{K}}}$ for the power context family $\vec{\mathbb{K}}$ in Fig. 19; as stated by (100), the k -ary relations are given by the columns of $\mathbb{K}_k$.

$$|\Delta| = \{1, 2, 3, 4, 5, 6\}$$

a	b	r	s
3	2	1 1	2 3
4	6	2 3	5 6
6		3 4	
		4 2	
		6 5	

$\mathbb{K}_1$:	a	b
1		
2		×
3	×	
4	×	
5		
6	×	×

$\mathbb{K}_2$:	r	s
(1,1)	×	
(2,3)	×	×
(3,4)	×	
(4,2)	×	
(5,6)		×
(6,5)	×	

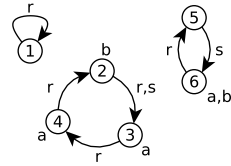


Fig. 18. Relational structure (drawn as tables).

Fig. 19. Power context family. **Fig. 20.** Intension graph.

Conversely, a power context family is obtained from a relational structure by arranging the k -ary relations in columns of $\mathbb{K}_k$. Relational structures and power context families thus contain the same information, and we can translate back and forth between them. The correspondence is not strictly one-to-one, because empty rows in a relation context of $\vec{\mathbb{K}}$ are not reflected in $\Delta_{\vec{\mathbb{K}}}$: however, no information is associated with such rows. Note that empty rows in the object context of $\vec{\mathbb{K}}$ state the existence of an object, and they are reflected in the carrier set of $\Delta_{\vec{\mathbb{K}}}$.

In some definitions, a power context family is a sequence $(\mathbb{K}_k)_{k \geq 0}$. Then $\mathbb{K}_0$ takes the role of the object context in Def. 19, and $\mathbb{K}_1$ is considered a relation context. In this case, the attributes of $\mathbb{K}_0$ and $\mathbb{K}_1$ both correspond to unary relations; a possible use is to reserve $\mathbb{K}_0$ for sort attributes.

7.2 Intension Graphs

We have seen in Figs. 18 and 19 that relational structures are obtained by organizing the information in a power context family column-wise. If the information is organized row-wise, we obtain a labeled graph (cf. Fig. 20). The rows of the object context correspond to labeled nodes, and the rows of the relation contexts correspond to labeled edges. Intension graphs, introduced below, are modeled after Wille's concept graphs, which are based on relational graphs [38, p. 167].

Definition 4. *An intension graph over a relational signature M is a triple (V, E, κ) , where V is a set of vertices (or nodes), $E \subseteq \bigcup_{k \geq 2} V^k$ is a set of edges, and κ is a function defined on $V \cup E$ with $\kappa(v) \in \mathfrak{P}(M_1)$ and $\kappa(e) \in \mathfrak{P}(M_k) \setminus \{\emptyset\}$ for all $v \in V$ and $e \in E^{(k)} := E \cap V^k$, $k \geq 2$.*

The intension graph (V, E, κ) associated with a given power context family $\vec{\mathbb{K}}$ (specified as before) is thus given by

$$V := G_1, \quad (101)$$

$$E^{(k)} := G_k, \quad (102)$$

$$\kappa(g_1, \dots, g_i) := (g_1, \dots, g_i)^{I_i} \quad (103)$$

for all $k \geq 2$, $i \geq 1$ and $(g_1, \dots, g_i) \in G_i$. Note that edge labels must be non-empty. The power context family can be reconstructed (except for empty rows in relation contexts) by translating labeled nodes and labeled edges into context rows. In summary, relational structures, power context families and intension graphs are equivalent data representations, and we can easily translate back and forth between them.

Throughout the paper, we have drawn conjunctive queries as graphs, while observing that tableau queries are more faithfully represented drawn as tables (cf. Section 3.2). By equipping intension graphs with a window, we obtain a representation of conjunctive queries that formally reflects the drawings.

Definition 5. *A windowed intension graph is a pair $(\mathcal{G}, \lambda)$ consisting of an intension graph $\mathcal{G}$ and a partial function $\lambda : \mathbf{var} \rightarrow |\mathcal{G}|$.*

A one-to-one correspondence between relational structures and intension graphs is easily established, e.g. via power context families (cf. Figs. 18, 19 and 20). A database can be modeled by an intension graph $\mathcal{D}$, and a query by a windowed intension graph $(\mathcal{G}, \lambda)$, and a homomorphism $f : \mathcal{G} \rightarrow \mathcal{D}$ is defined according to its counterpart for relational structures, i.e. as a function between the node sets which preserves edges and labels. The content of the previous sections can thus be stated equivalently on the basis of intension graphs, rather than relational structures. Details can be found in an earlier paper by Kötters [23].

The advantage of considering different formalisms is that they support different ways of thinking about the material, and facilitate different kinds of knowledge transfer and problem solving. For example, the relational structures formalism establishes a connection from Formal Concept Analysis to database theory and classical logic. The intension graph formalism may be more intuitive to use when working with graph theoretical properties.

We consider one further formalism, where queries are windowed intension graphs and data is represented by a power context family; this is useful for a comparison with concept graphs in the next section. A solution, as defined below, is the analog of a homomorphism of relational structures (or intension graphs) in this setting.

Definition 6. Let $\mathcal{G} = (V, E, \kappa)$ be an intension graph and $\vec{\mathbb{K}}$ a power context family over the same relational signature M . A solution of G in $\vec{\mathbb{K}}$ is a function $f : V \rightarrow G_1$, extended to E by $f(v_1, \dots, v_k) := (f(v_1), \dots, f(v_k))$, such that $f(v) \in \kappa(v)^{I_1}$ and $f(e) \in \kappa(e)^{I_k}$ for all $v \in V$ and $e \in E^{(k)}$, $k \geq 2$.

A solution can be considered a kind of homomorphism (between heterogeneous structures), so we write $f : \mathcal{G} \rightarrow \vec{\mathbb{K}}$ to denote that f is a solution of G in $\vec{\mathbb{K}}$. The result of the conjunctive query (G, λ) in $\vec{\mathbb{K}}$ is then

$$\text{res}_{\vec{\mathbb{K}}}(\mathcal{G}, \lambda) := \{f \circ \lambda \mid f : \mathcal{G} \rightarrow \vec{\mathbb{K}}\}, \quad (104)$$

cf. (12).

7.3 Concept Graphs

Several variants of concept graphs have been defined. We only consider *simple concept graphs* [38], corresponding to the first definition of concept graphs of Wille [36]. A simple concept graph over the power context family $\mathbb{K}$ is a 5-tuple $(V, E, \nu, \kappa, \varrho)$ consisting of a set V of vertices, a set E of edges, a function $\nu : E \rightarrow \bigcup_{k \geq 1} V^k$ that associates each edge with its (ordered sequence of) endpoints, a function κ that assigns a concept $\kappa(v) \in \mathcal{B}(\mathbb{K}_1)$ to each $v \in V$ and a concept $\kappa(e) \in \mathcal{B}(\mathbb{K}_k)$ to $e \in E^{(k)} := \{e \in E \mid \nu(e) \in V^k\}$, $k \geq 1$, and a realization $\varrho : V \rightarrow \mathfrak{P}(G_1) \setminus \{\emptyset\}$ such that

$$\varrho(v) \subseteq \text{ext}(\kappa(v)), \quad (105)$$

$$\varrho(v_1) \times \dots \times \varrho(v_k) \subseteq \text{ext}(\kappa(e)) \quad (106)$$

for all $v \in V$, $e \in E^{(k)}$, $k \geq 2$ and $(v_1, \dots, v_k) = \nu(e)$. We discuss this definition in the following, in the context of our work.

Abstract concept graphs were defined in the first paper on concept graphs [36], but not considered thereafter. In that paper, a concept graph without a realization is considered an abstract concept graph, more specifically an *abstract concept graph over the power context family* $\vec{\mathbb{K}}$, because its labels are concepts of contexts in $\vec{\mathbb{K}}$. In the above definition, we would thus consider the 4-tuple (V, E, ν, κ) an abstract concept graph over $\vec{\mathbb{K}}$. Such abstract concept graphs, where labels $\kappa(u)$ are formal concepts, are only a special case of abstract concept graphs; according to the general definition, a label $\kappa(u)$ must be an element of some set C of abstract concepts, but the notion of abstract concept is not further defined.

An intension graph is thus an abstract concept graph without multiple edges, where abstract concepts are sets of attributes (cf. Def. 4). Intension graphs with multiple edges were defined in Kötters [24], but some care needs to be taken with the definition of homomorphisms in the presence of multiple edges. Labeling nodes and edges by attribute sets (instead of concepts of a context $\mathbb{K}_k$, $k \geq 1$) makes intension graphs independent of the data, as would be expected for modeling queries.

However, an intension graph $\mathcal{G} = (V, E, \kappa)$ over M can be identified with an abstract concept graph over $\vec{\mathbb{K}}$ for any given $\vec{\mathbb{K}}$ over M , by identifying node labels $\kappa(v)$ and edge labels $\kappa(e)$ with the concepts $\tilde{\kappa}(v) := (\kappa(v)^{I_1}, \kappa(v)^{I_1 I_1})$ and $\tilde{\kappa}(e) := (\kappa(e)^{I_k}, \kappa(e)^{I_k I_k})$, respectively, for $v \in V$, $e \in E^{(k)}$ and $k \geq 2$. Then the definition of a realization in (105) and (106), applied to $\tilde{\kappa}$, is equivalent to

$$\varrho(v) \subseteq \kappa(v)^{I_1}, \quad (107)$$

$$\varrho(v_1) \times \dots \times \varrho(v_k) \subseteq \kappa(e)^{I_k}. \quad (108)$$

We can thus identify a pair $(\mathcal{G}, \varrho)$, where ϱ satisfies eqns. (107) and (108), with a simple concept graph (without multiple edges).

Now let us consider the meaning of eqns. (107) and (108). If $f : \mathcal{G} \rightarrow \vec{\mathbb{K}}$ is a solution, then $\varrho(v) := \{f(v)\}$ defines a realization (cf. Def. 6). Conversely, if ϱ is a realization, then every function $f : V \rightarrow G_1$ with $f(v) \in \varrho(v)$ is a solution. In short, a function $\varrho : V \rightarrow \mathfrak{P}(G_1) \setminus \{\emptyset\}$ is a realization if and only if the elements of the sets $\varrho(v)$ can be freely combined to obtain solutions.

The identification of unary concepts with nodes in connected components (and of connected components with systems of related concepts) in Section 5.5 suggests another notion of concept graph. In the current formalism, such a component is represented by an intension graph $\mathcal{G}$, a unary concept corresponds to a node v of $\mathcal{G}$, and its extent is the one-column table $\text{res}_{\vec{\mathbb{K}}}(\mathcal{G}, v)$. The function

$$\text{ext}_{\mathcal{G}}(v) := \{f(v) \mid f : \mathcal{G} \rightarrow \vec{\mathbb{K}}\} \quad (109)$$

describes the extents of all unary concepts in $\mathcal{G}$ as subsets of G_1 , and we call the pair $(\mathcal{G}, \text{ext}_{\mathcal{G}})$ a *projectional concept graph* (previously defined in [24]) if the sets $\text{ext}_{\mathcal{G}}(v)$ are non-empty (i.e. if $\mathcal{G}$ has a solution in $\vec{\mathbb{K}}$).

In general there are many simple concept graphs $(\mathcal{G}, \varrho)$ for given $\mathcal{G}$ and $\vec{\mathbb{K}}$, but the function $\text{ext}_{\mathcal{G}}$ is uniquely determined. Elements of the sets $\text{ext}_{\mathcal{G}}(v)$ can not be freely combined to obtain solutions, but each $g \in \text{ext}_{\mathcal{G}}(v)$ can be extended to some solution. Necessarily, $\varrho(v) \subseteq \text{ext}_{\mathcal{G}}(v)$ for all $v \in V$.

8 Related Work

Power context families and concept graphs are introduced in Wille [36]. Concept graphs are presented as a mathematical formalization of Sowa's Conceptual Graphs [33] consistent with a Formal Concept Analysis treatment. Different kinds of concept graphs are presented in Wille [38] but, to the knowledge of the authors, abstract concept graphs are mentioned only in the introductory paper by Wille [36] and are the only kind of concept graphs defined without a realization.

Windowed structures were introduced as triples $(X, \nu, \mathcal{G})$ in Kötters [21], and the core content of Section 5, together with a family relations example, is presented there. A subsequent paper by Kötters and Schmidt [25] addresses the connection to logic, deals with sorts, uses “augmented structures” to represent values in databases and shows the connection to pattern structures by representing extensions as sets of partial interpretations. An adaptation of the conceptual model to RDF data is described in Kötters [22]. A navigation application on the basis of conjunctive query concepts, and the applicability to databases in general, is described in Kötters [24].

Pattern structures are introduced in Ganter and Kuznetsov [15], and the use of Conceptual Graphs as patterns is also suggested there. The representation of conjunctive queries (and thus primitive positive formulas) by graphs, and of entailment by graph homomorphism, is credited to Chandra and Merlin [7].

Baader and Molitor [5] describe concept lattices where intents are descriptions in some description logic. Different logics have been considered, and structural representations of descriptions using trees and homomorphisms [4] or graphs and simulations [2] are used to compute least common subsumers as products. In particular, descriptions in the logic $\mathcal{EL}_{\text{gfp}}$ correspond to conjunctive queries represented by rooted trees, as obtained by the unraveling operator in Section 6.2. While descriptions may contain cycles, these are interpreted in the sense of unfolding/unraveling; the semantics of Description Logic being different from that of classical logic.

Relational Concept Analysis (RCA) allows one to compute concept lattices from relational context families, which are similar to power context families but distinguish between objects of different sorts. The meaning of concept descriptions is parameterized by scaling operators; conjunctive queries are obtained if the existential scaling operator is used but, as for Descriptions Logics, intents correspond to rooted trees. Relational Context Families (RCA) and the construction algorithm are described in Huchard et. al [20]. A more detailed comparison of RCA with conjunctive query concepts is not presented here but can be found in Kötters [23].

Liquiere [27] and Kuznetsov [26] describe concept lattices with sets of labeled graphs as intents. This approach was used in various pattern mining settings, from chemoinformatics to mining UML diagrams.

Ferré [12] uses projected graph patterns as concept intents. These correspond to conjunctive queries, and their extent corresponds to a result table in the unnamed perspective (cf. [1, Section 3.2]). The lattice of n -ary concepts, defined by Ferré for each arity $n \geq 0$, corresponds to the lattice $\mathfrak{L}_\Delta[\{x_1, \dots, x_n\}]$ of this paper (for any n variables $x_1, \dots, x_n$).

In Chein and Mugnier [8], λ -BGs are presented, which are Basic Conceptual Graphs with distinguished concepts given by a function λ , and these express conjunctive queries (our use of λ in this paper derives from this treatment). Moreover, in Chein and Mugnier [8, Chapter 8], the categorical product is used to describe the least generalization of Conceptual Graphs. Rossman [31] also describes comparable product and co-product constructions for structural representations of primitive positive formulas in a model theoretic context.

9 Conclusion

In this paper concept lattices with conjunctive query intents are presented as a special case of Ganter and Kuznetsov’s pattern structures. Kernel operators for conjunctive queries were also introduced to reduce the complexity of the computation and/or simplify the description of concepts. While patterns are originally described by windowed structures, which generalize tableau queries to infinite queries, alternative formalisms, based on intension graphs and power context families, are introduced to further connect Formal Concept Analysis, database theory, model theory, logic (including description logic), Wille’s concept graphs, Sowa’s Conceptual Graphs, and by extension potentially to other fields of knowledge representation.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Baader, F.: Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In: Gottlob, G., Walsh, T. (eds.) Proceedings of IJCAI 2003. pp. 319–324. Morgan Kaufmann (2003)
3. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press (2017)
4. Baader, F., Küsters, R., Molitor, R.: Computing least common subsumers in description logics with existential restrictions. In: Dean, T. (ed.) Proceedings of IJCAI 1999. pp. 96–101. Morgan Kaufmann (1999)
5. Baader, F., Molitor, R.: Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In: Ganter, B., Mineau, G.W. (eds.) Proceedings of ICCS 2000. LNCS, vol. 1867, pp. 292–305. Springer, Berlin, Heidelberg (2000)

6. Birkhoff, G.: Lattice Theory. American Mathematical Society, Providence, 3rd edn. (1967)
7. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. In: Proceedings of the ninth annual ACM symposium on Theory of computing. pp. 77–90. STOC '77, ACM, New York, NY, USA (1977)
8. Chein, M., Mugnier, M.L.: Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs. Advanced Information and Knowledge Processing, Springer, London (2009)
9. Ebbinghaus, H.D.: Einführung in die Mengenlehre. Spektrum Akademischer Verlag, Heidelberg, Berlin, 4 edn. (2003)
10. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic. Springer, New York, second edn. (1994)
11. Ern , M.: Einf hrung in die Ordnungstheorie. B.I.-Wissenschaftsverlag, Mannheim (1982)
12. Ferr , S.: A proposal for extending formal concept analysis to knowledge graphs. In: Baixeries, J., Sacarea, C., Ojeda-Aciego, M. (eds.) Formal Concept Analysis - 13th International Conference, ICFCA 2015, Nerja, Spain, June 23-26, 2015, Proceedings. LNAI, vol. 9113, pp. 271–286. Springer (2015)
13. Frege, G.:  ber Sinn und Bedeutung. Zeitschrift f r Philosophie und philosophische Kritik 100, 25–50 (1892)
14. Ganter, B.: Two basic algorithms in concept analysis. In: Kwuida, L., Sertkaya, B. (eds.) Formal Concept Analysis. pp. 312–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
15. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Proceedings of ICCS 2001. LNCS, vol. 2120, pp. 129–142. Springer (2001)
16. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer, Berlin (1999)
17. Godin, R., Saunders, E., Gecsei, J.: Lattice model of browsable data spaces. Inf. Sci. 40(2), 89–116 (1986)
18. Groh, B., Eklund, P.W.: Algorithms for creating relational power context families from conceptual graphs. In: Proceedings of ICCS 1999. pp. 389–400 (1999), https://doi.org/10.1007/3-540-48659-3_24
19. Groh, B., Strahringer, S., Wille, R.: Toscana-systems based on thesauri. In: Mugnier, M.L., Chein, M. (eds.) Conceptual Structures: Theory, Tools and Applications. pp. 127–138. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
20. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Annals of Mathematics and Artificial Intelligence 49(1-4), 39–76 (2007)
21. K tters, J.: Concept lattices of a relational structure. In: Pfeiffer, H.D., Ignatov, D.I., Poelmans, J., Gadiraju, N. (eds.) Proceedings of ICCS 2013. LNCS, vol. 7735, pp. 301–310. Springer (2013)
22. K tters, J.: Concept lattices of rdf graphs. In: Ojeda-Aciego, M., Baixeries, J., Sacarea, C. (eds.) Proceedings of the International Workshop on Formal Concept Analysis and Applications 2015. CEUR Workshop Proceedings, vol. 1434. CEUR-WS.org (2015), <http://ceur-ws.org/Vol-1434>
23. K tters, J.: Intension graphs as patterns over power context families. In: Huchard, M., Kuznetsov, S. (eds.) Proceedings of CLA 2016. CEUR Workshop Proceedings, vol. 1624. CEUR-WS.org (2016)

24. Kötters, J., Eklund, P.W.: The theory and practice of coupling formal concept analysis to relational databases. In: Kuznetsov, S.O., Napoli, A., Rudolph, S. (eds.) Proceedings of FCA4AI 2018. CEUR Workshop Proceedings, vol. 2149, pp. 69–80. CEUR-WS.org (2018)
25. Kötters, J., Schmidt, H.W.: A database browser based on pattern concepts. In: Carpineto, C., Kuznetsov, S.O., Napoli, A. (eds.) Proceedings of FCAIR 2013. CEUR Workshop Proceedings, vol. 977. CEUR-WS.org (2013), <http://ceur-ws.org/Vol-977>
26. Kuznetsov, S.O.: Learning of simple conceptual graphs from positive and negative examples. In: Żytkow, J.M., Rauch, J. (eds.) Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '99). LNAI, vol. 1704, pp. 384–391. Springer (1999)
27. Liquiere, M., Sallantin, J.: Structural machine learning with galois lattice and graphs. In: Proc. of the 1998 Int. Conf. on Machine Learning (ICML'98). pp. 305–313. Morgan Kaufmann (1998)
28. Maier, D., Tekle, K.T., Kifer, M., Warren, D.S.: Declarative logic programming. chap. Datalog: Concepts, History, and Outlook, pp. 3–100. Association for Computing Machinery and Morgan & #38; Claypool, New York, NY, USA (2018), <https://doi.org/10.1145/3191315.3191317>
29. Mill, J.S.: A System of Logic: Ratiocinative and Inductive (2002)
30. Peirce, C.: The Collected Papers of Charles Sanders Peirce, vol. 4 (1958)
31. Rossman, B.: Homomorphism preservation theorems. J. ACM 55(3), 15:1–15:53 (Aug 2008), <http://doi.acm.org/10.1145/1379759.1379763>
32. Saada, H., Huchard, M., Liquiere, M., Nebut, C.: Learning model transformation patterns using graph generalization. In: Bertet, K., Rudolph, S. (eds.) Proceedings of CLA 2014. CEUR Workshop Proceedings, vol. 1252, pp. 11–22. CEUR-WS.org (2014)
33. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley (1984)
34. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In: Rival, I. (ed.) Ordered Sets, pp. 445–470. Reidel, Dordrecht–Boston (1982)
35. Wille, R.: Restructuring mathematical logic: an approach based on Peirce's pragmatism. In: Ursini, A., Aglianò, P. (eds.) Logic and Algebra, pp. 267–281. Marcel Dekker, New York (1996)
36. Wille, R.: Conceptual graphs and formal concept analysis. In: Lukose, D., Delugach, H.S., Keeler, M., Searle, L., Sowa, J.F. (eds.) Proceedings of ICCS 1997. LNCS, vol. 1257, pp. 290–303. Springer, Heidelberg (1997)
37. Wille, R.: Concept graphs as semantic structures for contextual judgment logic. In: Eklund, P., Diatta, J., Liquiere, M. (eds.) Proceedings of CLA 2007. CEUR Workshop Proceedings, vol. 331, pp. 3–12. CEUR-WS.org (2007)
38. Wille, R.: Formal concept analysis and contextual logic. In: Hitzler, P., Schärfe, H. (eds.) Conceptual Structures in Practice, pp. 137–173. Chapman & Hall/CRC (2009)